THE UNIVERSITY OF ALBERTA

AN INTERROGATIVE AUTHORING SYSTEM

BY

(C) ANNABELLE YASGUL PALOIAN

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1971

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "An Interrogative Authoring System" submitted by Annabelle Y. Paloian in partial fulfilment of the requirements for the degree of Master of Science.

Date... July 6/71 .....

ABSTRACT

An investigation was made into the problems encountered in CAI authoring that are posed by present authoring systems. Some requirements for a desirable authoring system have been reviewed and a proposal presented for an interrogative authoring system.

The major objectives of the study were: (a) to design an authoring system that facilitates course authoring for the instructor, (b) to develop an authoring language composed of natural English language words that are semantically compatible with the teacher's occupational vocabulary, (c) to create a list structural representation of each language element as opposed to compiling each statement and generating code, and (d) to use the list structural representation in the generation of a course in machine code. The system has been designed to facilitate the generation of any number of suitable machine code representations from the list structure.

A symbolic language has been designed as input to the authoring system and specified in detail as to construction and use. A list structure representation has been established for each symbolic language element. The design of each system component is reviewed and specific details concerning implementation presented. A survey of the present design is presented and the system evaluated. Finally, further extensions and improvements to the system design have been suggested.

ACKNOWLEDGEMENTS

I would like to express my appreciation to Dr. Steve Hunka for his patience, guidance, and advice throughout the duration of this research. Special thanks are extended to Norman McGinnis of the Division of Educational Research Services for many hours of informative discussion on CAI and the 1500 system. Finally, I wish to thank my Friend, who supplied hope.

The Department of Computing Science provided the financial assistance and facilities which made this project possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 A Brief Description of CAI

Several generations of computers have evolved since the original vacuum tube computer. Each generation of computer has had its defined set of capabilities, and a larger range of applications dependent upon the software constructed for it. Computers exist today with the hardware and software capable of allowing them to be used in a time-sharing and interactive mode. The latter two characteristics have made available some rather unique applications.

The field of education, in particular, has drawn upon the resources of computer technology in the development of computer assisted instruction (CAI). Instruction with the assistance of computers consists of a process which begins with the author and ends with the student in a learning environment. The author is usually an instructor or a subject-matter specialist who is knowledgeable in his own discipline, understands the content structure of the subject, as well as the characteristics of the students who are to be taught. The subject-matter and the techniques used to present the course are represented by a computer program, generally referred to as a course. The author must initiate the course by defining precisely the subject-matter to be taught and how it is to be taught. He may interact directly with the computer by testing and making corrections in the course material. He is always involved with the ongoing evaluation of the course as well as its final evaluation prior to giving it his approval for use by students.

The student learning phase consists of a number of students interacting with the subject-matter as it is represented in the computer as a program. The student interacts with the computer-presented course at a learning station. These stations usually consist of an output device with one or more media of input. Two of the devices that have been used for output are typewriter terminals and cathode ray tubes. Sometimes a combination of several output media may be used to present the course to the student. Response or input to the course is usually made through the use of a light pencil or a typewriter keyboard. Both modes of input may be available to the student.

CAI is still in the research and development stage with many problems to overcome before its potential of education can be adequately applied. These problems can be classified into three broad categories: the cost effectiveness and adequacy of hardware; the transition of an existing work force to a new concept in teaching; and the lack of efficient and flexible software to adequately interface the man-machine relationship.

The hardware problem is being dealt with by various equipment manufacturers who have recognized the potential of CAI systems for the educational market. The second problem is mainly one of providing for an acceptable and efficient transition of an existing work force from the traditional role of teaching to that by CAI. This transition will not occur rapidly nor smoothly. However, the traditional role of an instructor is undergoing a great deal of review, as it has been realized that many aspects of this role can be assumed by the computer. The lack of adequate software to interface the author to the machine has been the major concern of the writer and will be dealt with in more detail.

1.2 Author-Machine Interface

Within the field of CAI, there are at least three categories of personnel who participate in course construction (Tartar et al. 1968). They are as follows:

1. The subject-matter specialist
2. The instructional programmer
3. The coder

These three categories of personnel in practice have overlapping responsibilities and in the current developmental phase of CAI, all three roles may be assumed by one person. This hierarchy closely resembles that which exists in most fields of application of computers - the engineer, businessman or scientist who has a problem to be solved, the analyst or programmer who establishes a clear definition of the problem in mathematical or natural language, and the coder who translates this definition into a sequence of instructions in a language a computer can follow.

Ideally, in CAI, as in business or engineering, it should be the subject-matter specialist who instructs the computer directly in a language meaningful to him. Unfortunately, the languages computers can follow are often awkward to learn and use - hence the need for the other two categories of programmer and coder. Either the subject-matter specialist has to waste much time learning a computer-oriented language or a coder has to be trained in the art of teaching.

Even if this is accomplished, courses written in existing languages such as CW II (IBM/67) require a long period of debugging which is concerned primarily with essentially trivial computer-oriented

details. This situation requires the subject-matter specialist (or author) to devote such an excessive amount of time to producing a course that he may not bother with CAI.

Also because of the strong influence of the particular computer on the language used, the courses produced for one machine may not be usable on another, which makes the economics of writing courses even less attractive.

Figure 1.1 illustrates the interaction currently required between a computer and the author to produce a course.

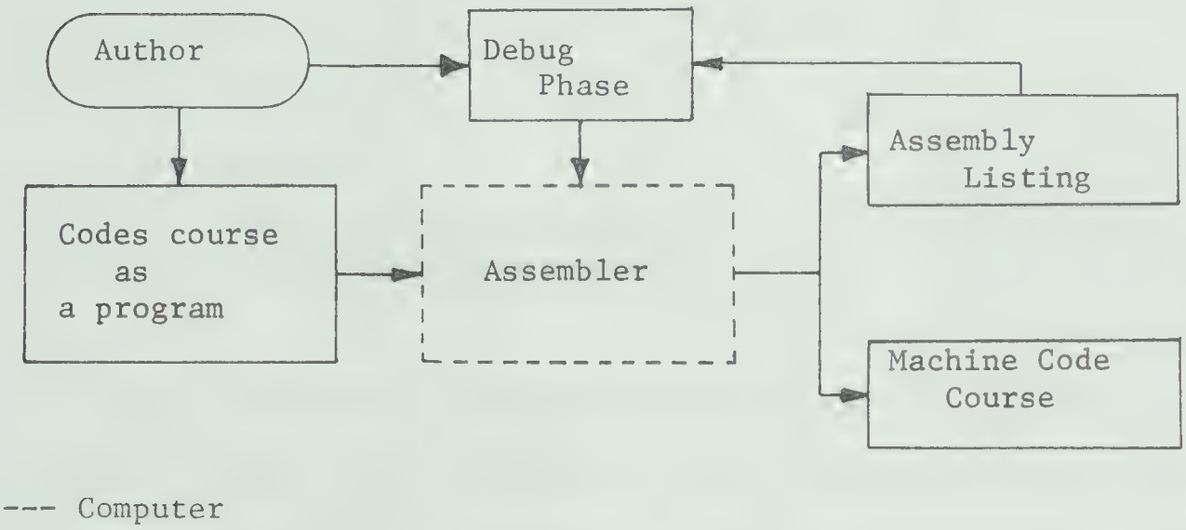

Figure 1.1 Current Author-Computer Interaction

The participants together with the tasks accomplished in Figure 1.1 are generally referred to as an "authoring system". It is to be noted that in the interaction illustrated, the author deals directly with the computer in the actions necessary to produce a course.

An authoring system has been designed in this thesis which provides for an interface between the author and the computer. Figure 1.2 illustrates the interaction required by the designed system.

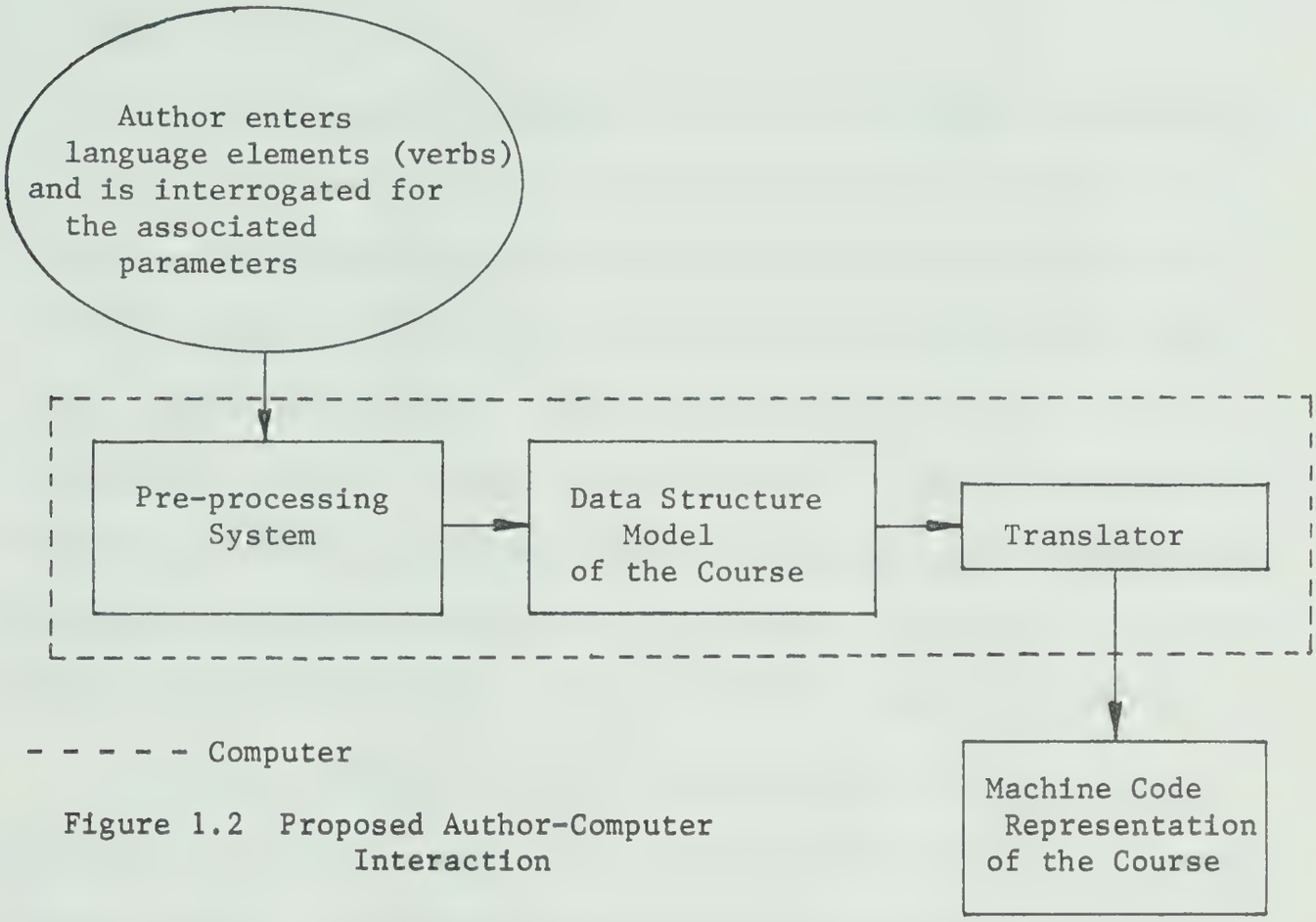

Figure 1.2 Proposed Author-Computer Interaction

The requirements for an authoring language and system are discussed in Chapter II. Included in this chapter is a proposal for an authoring system that involves the use of a pre-processor. Chapter III discusses the design of a symbolic authoring language and includes the language description. A list structure model is established in Chapter IV. Chapter V discusses the design of the pre-processing system. The specifications and techniques used in developing the pre-processing programs are presented in Chapter VI. The last chapter presents an evaluation of the system and discusses implications for future research.

CHAPTER II

AUTHORING SYSTEM REQUIREMENTS

2.1 Introduction

An authoring system consists of two major components, an authoring language and the operating system necessary to implement this language. These components are part of a communication scheme which involves a computer and a man, with the authoring system functioning as the link between the two. Therefore, the specifications of the communicants, namely, the man and the computer, directly determine the requirements of the authoring system. In addition, each component also influences the design of the other. Therefore, a complex specification scheme exists involving all "parties" of the communication system.

The requirements for an authoring language are presented in section 2.2 Section 2.3 discusses the requirements of an operating system for CAI. A proposal for a pre-processing system is presented in section 2.4.

2.2 Language Requirements

The purpose of an authoring system is to permit the author (instructor) to present to the computer a learning paradigm complete with subject-matter. This is presented in the form of a program which is executed by the computer for the student. Thus, there is created an individualized learning environment that expresses the instructional goals of the instructor. The requirements of an authoring language will therefore be considered from the points of view of the parties involved in its' use, namely, the author, the student, and the hardware.

2.2.1 The Author's Requirements

The primary task of an author should be instructional. Today however, most authors have had to concentrate on the technical and mechanical details of programming due to the demands of present languages and because of manpower allocations. Therefore, perhaps the first requirement an author would stipulate for an authoring system would be generally stated as 'ease of use'.

More specifically, the language should be ease to master and use. This requirement can be achieved by incorporating a wide range of principles in the design of the language. In particular, a programming language should have a small number of concepts that are general and systematic (Dijkstra 1963). It has been further stated that the presence of implicit conditions complicates the use of the language and does not contribute to its power (Tonge 1968).

The author also needs a language which is semantically compatible with the general and specific vocabulary that is used to represent the subject-matter and instructional strategies. Furthermore, the semantic relationships among the language elements (verbs) should conform to the author's own relational concepts. Imposing this requirement on the language, of course, immediately disposes of any symbolism or parameters that resemble coding in appearance.

Thus, one objective that would facilitate ease of use is meaningful representation of the language verbs. Not only would this facilitate course writing, but it would also tend to make the course language self-documenting, a feature much desired by authors and all programmers alike.

The choice of language verbs and their semantic representation is also influenced by the objectives of the author. To the author, the language is a means of constructing personalized teaching techniques. The realm of learning theory or instructional strategy and that of subject-matter content play a very important part in determining the functions of the language verbs. However, many problems arise as one studies existing instructional strategies. The techniques of teaching are in most cases either poorly defined or not defined at all.

Some teaching strategies have been recognized (Zinn-1 1968) and are as follows:

1. Drill and practice
2. Tutorial and testing
3. Conversational dialogue
4. Simulated environment
5. Competitive games

The language verbs in combinations must be capable of representing these strategies and also allow the author to represent individualized strategies. From a study of existing languages (Zinn-2 1968), it has been observed that the following basic capabilities must exist in an authoring language to meet the author's needs:

1. Display material
2. Accept response
3. Process response
4. Record and manipulate data for the author's decision strategy

5. Provide arithmetic operations for scorekeeping purposes

The capability of response processing, in particular, imposes a series of important constraints upon the language. The author needs 'good answer processing capabilities'. Included in this requirement is the capability of accumulating a past history of the student and then having the potential of individualizing instruction upon analysis of this data. This would be possible with convenient conditional expressions.

The language must also provide answer processing capabilities which are not restricted to 'right and wrong' analysis of replies. In many cases, a reply is neither right nor wrong and the author must have the flexibility of branching upon a class of replies rather than upon correct or incorrect analysis.

Thus, characteristics of various answer analysis techniques must also be considered in the design of the language verbs to ensure maximum flexibility at answer processing on the part of the author.

The aforementioned language capabilities are present in most authoring languages with varying degrees of sophistication depending upon the power of the language. Although this set of capabilities does satisfy a general type of user, it is not sufficient for all users. However, an author can construct most defined strategies within this defined functional framework.

Finally, in order to allow the author freedom in constructing new strategies or expressing personalized techniques, it would be desirable to design the language in a manner which would facilitate expansion.

2.2.2 Requirements of the Student

The needs of the student also contribute to the requirements of the language. The presented course, a result of the combination of language verbs, must create an environment conducive to learning. As the design of verbs concerned with the structure of the presentation must be influenced by the learning habits of the student, a course that is structured in a linear, and stereotyped manner can create an attitude of boredom and disinterest. Therefore, theories of educational psychology must also be considered in the language design. In addition, the student must have access to dictionaries and tables of information for reference purposes. The ability to compute within the context of the course is also desirable. Finally, some degree of course control must be entrusted to the student. This would involve the capability of transferring to sections within the course and initiating various learning sequences.

2.2.3 Hardware Considerations

The form of many authoring languages has been restricted by the computer hardware. The functions of input-output for example, are quite machine dependent in present day languages. Coursewriter II (IBM 1967) is one language in which the response media must be explicitly stated in the response request instructions. In addition, there are separate instructions for accepting responses from a typewriter keyboard and those from a light pencil.

As a course is predominantly concerned with the presentation of subject-matter, and request and analysis of responses, a greater degree

of freedom could by achieved by allowing the response media to be a variable and specified at execution time. This feature could be employed to vary the mode of course presentation and type of response media depending on a class of previous replies.

In conclusion, machine dependent languages are of benefit in applications which utilize the hardware capabilities. However, within the framework of CAI, dependency is a detriment rather than a benefit, as a machine dependent language produces a machine dependent course.

2.3 Operating System Requirements

The operating system consists of the software necessary to implement the language. Implementation usually encompasses the tasks of assembling, testing, evaluating, and executing the course. The design of this software must reflect the requirements of the parties involved in these designated tasks, namely, the author, the language, and the hardware.

2.3.1 The Author's Requirements

The author is involved with the operating system in the phases of programming, modifying, testing, and evaluating the course. First of all, the requirement 'ease of use' must be considered in the design of this system. This requirement is especially important in the modification and testing phases of a course. Presently, most operating systems consist of an assembler or compiler which is used to generate the machine code course. Programming, testing, and modifying a course therefore, involves a task that is technical in nature and not natural to a non-programmer. Ideally, the operating system should facilitate these tasks by interacting with the author in his natural language. In

addition, meaningful debugging tools such as functional flowcharts and error checking pre-compilers should also be available in the system.

2.3.2 Language Design

The characteristics of an operating system depend also on the design of the language. A desirable feature in a language is the capability of extending its power. Therefore, the system must also be designed for expansion. Table driven software would accomplish this goal.

The author must also be allowed control of some of the hardware features, in particular, control of timing of course presentation.

The design of this system should also ensure machine independence of the language elements. In particular, the functions of input and output could be treated as variable. The operating system should be able to support this feature.

2.3.3 Hardware Implications

As CAI is only economically feasible with hardware that is capable of being used in a time-sharing environment, certain special requirements must be specified for the operating system.

It is often advantageous to study requirements of existing systems in determining those of new ones. Maestro is a system developed by Silvern (Silvern 1966). It uses Lyric as the authoring language. One of the specifications made is the following: "... as the system should handle as many students simultaneously in a time-sharing environment, re-entrant coding should be used." The following considerations lead to the development of a CAI system at the University of California at Irvine (Tonge 1968):

1. "The system should permit the entry of course programs either on-line or using batch input devices and should allow the author to correct and immediately test material on-line, even during student usage of other parts of the same course."

2. "The system should provide to the course author capabilities for calculation, text analysis, data base access, and abbreviated reference to commonly used sequences of material."

3. "The system should allow "easy" modification of language syntax and semantics (by staff programmers) so as to encourage course authors to consider and suggest language improvements."

4. "The system should provide a computation capability to students, in the context of the course program."

5. "The system should be implemented so as to minimize the cost, while processing student interactions, of the authoring capabilities."

Therefore, designing a good authoring system is a matter of considering what the langauge is to do, considering the requirements of the user, and the hardware environment within which the system is to operate.

In most cases, authoring system design has taken into consideration the hardware environment and the task to be accomplished, at the expense of the user's requirements. There are many solutions to this problem. One is to design a special authoring language and system for various classes of users, depending upon their requirements. An alternative solution is to maintain the present authoring sytem and hire a programmer. This introduces another individual who is responsible for transmitting to the computer, the intentions of the author. Another approach involves the introduction of a pre-processor into the authoring system. Its function would be to interface the authoring language and the operating system, or more specifically, interface the author and the computer.

The latter approach has been chosen and will now be described.

2.4 A Pre-processing System

2.4.1 A Proposal

Basically, the pre-processing system is responsible for author interaction during the stages of entering, modifying, testing, and evaluating a course. It consists of a symbolic authoring language, a pre-processor, and a multi-lingual translator. The design of this system has been influenced by the requirements stated in sections 2.2 and 2.3 of this chapter.

The proposed authoring system consists of three separate stages. The first stage involves the author, the symbolic language, and the pre-processor. Authoring is conducted in an interactive and time-sharing environment. The author selects elements (verbs) from the language to create the course. These verbs are then entered individually. The pre-processor has two tasks. Upon validating the entered verb for correctness, it interrogates the author for information (parameters) related to the verb. The author is responsible for providing these parameters upon request. The pre-processor then creates a data structure representation of the verb and its parameters. The author may switch to edit mode and modify the course at any time during authoring. Modification would be to the data structure representation. Meaningful flow pattern of the course can be produced from this structure representation to aid the author in debugging. When the course is fully entered and correct, the next stage may begin.

The second stage involves the data structure representation produced by stage one and the multi-lingual translator. The translator

is to be designed to produce an existing authoring language code for the data structure representation. This code is then used as input to the third and final stage which involves this code and the operating system.

The third stage is the actual execution of the course by the operating system. This system consists of the software needed to support the resulting authoring language code produced by stage two.

Of this entire authoring system, stage one, the symbolic language and pre-processor have been developed in this thesis. The task of the multi-lingual translator is a straightforward transformation process and has not been described. Stage three, the operating system or support software, exists for codes that are to be produced by the translator.

One of the objectives of this system is to generate a program from the data structure representation of the course. An attempt was made by IBM to develop a course generating authoring system. CG-1, "A Course Generating Program for CAI" (Meadow et al. 1968), was developed to converse with the author in natural language and generate a course from the data gathered in the conversation. The program and course generated, were written in the PL/I programming language. Although the project had great potential, it was not completed. The CG-1 concepts of course generation and conversational gathering of data have influenced the design of the pre-processing system.

2.4.2 Unique Aspects

There are several unique aspects of this type of authoring system. In particular, the authoring language demands very little

detailed specification on the part of the author. The few natural language parameters associated with the verb are requested of the author by the system, thus eliminating the possibility of missing or erroneously specified parameters. The technique of interrogation, as used in this system, ensures a syntactically correct representation of the verb since the pre-processor and not the author is responsible for constructing the input to the translator.

Another unique aspect of the pre-processor is the use of a data structure representation of the course or program. This structure can easily lend itself to many tasks such as a data source for debugging, course analysis, and evaluation, but more important, as input to the multi-lingual translator. This is one approach to the problem of course sharing mentioned in Chapter I. With this approach, one representation of the course can exist with many language translations generated for this representation.

Finally, it is to be noted that the operating system is responsible only for executing the course rather than providing for the complete author-machine interface.

CHAPTER III

THE LANGUAGE

3.1 Introduction

The language that has been designed for the pre-processor can be classified as a symbolic language. By definition, a symbol is "something that stands for something else". In this language, the verbs are words, chosen from a subset of the natural English language. To the CAI author, they stand for instructional actions (functions). Consequently, the syntax of the language verbs is informal or non-existent when considered in the context of most other programming languages.

The rationale used for the language design is presented in section 3.2. Section 3.3 contains a description of the language. The detailed aspects of language use are presented in section 3.4. Section 3.5 discusses the use of groups of verbs (Patterns).

3.2 Design Considerations

The first area of consideration in authoring language design is that of language structure. The simplest method to structure a program is, of course, to provide the capability of labeling verbs. This allows branching to these verbs, which usually mark the beginning of a certain new section of code. This technique keeps the program modular and facilitates modification.

All authoring languages provide a framework within which the language verbs are used. Writeacourse, an authoring language recently developed at the University of Washington (Hunt et al. 1968), defines the course as a Lesson, within which are STATEMENTS. These STATEMENTS

are composed of INSTRUCTIONS, which in turn are composed of VERBS. CAL, developed at the University of California (Tonge 1968), defines the course as a series of CHAPTERS. A CHAPTER is composed of LINES.

Structure design is critical as it can impose a rigid framework on the course. Within such a rigid framework, CAI can easily be turned into programmed instruction, where the computer becomes a simple page presenting device.

A simple framework or structure has been chosen for the symbolic language. In a sense, it is procedure oriented in that the concept of BLOCKS is used. The BLOCK or unit, starts with the verb BEGIN, which is to be labeled and ends with the unlabeled verb END. A course can consist of numerous BLOCKS, or just a single BLOCK. Nesting of BLOCKS however, is not allowed.

The language verbs themselves are to be used within a BEGIN-END sequence and represent the capabilities required by the author. These capabilities can be categorized into five different types. The functions, which represent several aspects of the capabilities, are listed under each type. The categories and functions are as follows:

1. Display subject-matter
    (a) Display textual information
    (b) Display a question
    (c) Display a choice
2. Accept response

    Implicitly assumed in the following functions:
    (a) Display a question
    (b) Display a choice

3. Analyze response
    (a) Identify a single predictable response
    (b) Identify the presence of a given set of predictable response (AND condition)
    (c) Identify the presence of any member of a given set of predictable responses (OR condition)
    (d) Identify the presence of predictable keywords in the response
    (e) Identify the selection of a choice
    (f) Identify unpredictable responses

4. Sequence the presentation of a course
    (a) Control timing of course presentation
    (b) Provide for conditional and unconditional branching

5. Record and manipulate data for author's use
    (a) Record the identification of predictable responses
    (b) Provide branching control upon analysis of recorded data
    (c) Provide capability to record path traversed

6. Arithmetic operations
    (a) Establish and set counters
    (b) Add to counters
    (c) Subtract from counters

Although this set of functions will not accomplish every author's objectives, it is sufficient for a general class of user and may be expanded upon unanticipated author needs.

The main objective in the design of a symbolic terminology is, of course, to choose semantically meaningful word representations for each function. As several authoring languages have been designed with this objective, a comparative study of these and several programming languages would prove useful in establishing a common terminology for each of the listed functions. The following discussion will center on around each of the categories just mentioned.

A wide variety of terminology has been used for the display of subject-matter, or, more specifically, for output of information to the student. In most cases, the resulting verb terminology is related to the output device. The Writeacourse system uses a typewriter terminal as the input-output media. The verb PRINT is used to display both information and questions with no differentiation between the two functions. The TUTOR language is used at the University of Illinois (Avner et al. 1969). The output verb, WRITE, similar to that used in Fortran, is used to present both questions and information. Course-writer II, uses the mneumonic TY for TYPE and DT for DISPLAY TEXT. Another language, Vault (Romaniuk 1970), developed at the University of Alberta, comes closer to machine independence in its' terminology than the others discussed. The name of the function is meaningful in terms of the functional objective rather than this objective and the associated hardware. The verb DISPLAY is used for output of information and the verb QUESTION to display a question. However, the parameters associated with these two verbs confine the use of the language to systems

utilizing a CRT display for input and output of information to the student.

In this design it was considered desirable to choose terminology that is both meaningful in relation to the author's needs and CAI function required, and semantically machine independent. Consequently, the verb TEXT has been chosen to represent the presentation of information and the verb ASK to define the function 'present a question'.

A special verb has been designed that allows the author to present multiple choices to the student. This verb has been included since a choice is not usually a question although it does initiate a response. Therefore, an ASK would not be semantically appropriate for this function. The verb CHOICE allows facts or questions to be presented in a format suitable for selection by the student. In most cases, a multiple choice strategy can be represented by using an ASK to present the question and several CHOICE verbs to present the selections related to the question.

Almost all authoring languages have separate verbs for response acceptance. These are used in conjunction with the verbs that present questions. Writeacourse has one such verb called ACCEPT, which is directed to the input media, i.e. the typewriter terminal. FOIL (Hesselbart 1968), also uses the verb ACCEPT.

The Vault language provides a display verb QUESTION which implicitly sets up the response request from the typewriter terminal while other verbs POINT and DISPLAY INSERT are used in conjunction to present light pen choices and accept responses.

In an attempt to divorce the response media from the response request, verbs representing the category 'accept response' have not been designed in the symbolic language, but rather this function is assumed to be directly a part of the display of a question or choice.

Response analysis is an area in which great power and flexibility are required by the author because the domain of responses available to a student is extremely large. Alternatives to providing powerful functions for response analysis are usually tedious, lengthy and redundant codings which require frequent updating. In particular the flexibility to analyze responses for keywords and strings of words must be incorporated in the analysis verbs.

Writeacourse provides the verb CHECK to identify an exact response and CHECKIN to match on keywords. TUTOR has an extensive list of verbs for response analysis. ANS is used to define correct answers. A list of characters can be ignored in the answer by using BUMP. A mandatory sequence of responses is specified by the verb MUST. Finally, the verb SPELL is provided to check the student's response for possible misspellings of the author's ANS command.

Generally speaking, the conditional expression IF ( ) THEN ( ) has had widespread use in authoring and programming languages for powerful analysis purposes. The brackets are to be replaced by parameters. Therefore, the expression IF ( ) THEN ( ) has been chosen as the basis for the design of answer processing verbs. Variations of this expression have been designed to denote the various answer processing functions. The basic expression IF ( ) THEN ( ) can be used to identify a single response or a group of responses. The variation

IF ANY ( ) THEN ( ) is used to identify sentence type responses or the presence of a set of responses. Thus, the AND and OR conditions are represented. Keywords are identified through the use of the expression IF KEYWORD ( ) THEN ( ). Selection of a choice can be recognized through the use of the verb IF CHOICE ( ) THEN ( ).

Coursewriter II provides a verb UN for unrecognizable replies. It is the only authoring language that does so. The negative or ELSE branch of the conditional expression IF ( ) THEN ( ) ELSE ( ) is usually sufficient to handle this condition. As the conditional expression used in the symbolic language does not contain the ELSE parameter, the next instruction in sequence is executed if the condition specified by the expression is found to be false.

Course sequencing usually consists of presentation timing and the use of conditional and unconditional verbs for branching. In most authoring systems, direct control of the hardware is not possible or has not been provided for. Two languages, Coursewriter II, and Vault, however, do provide this feature through the use of the verb PAUSE. Although this function may not prove useful to all authors, it is included for the sake of maximum flexibility. The verb designed for this purpose is DELAY.

The standard verb for unconditional transfer is GO TO and has been adopted with no modification.

Finally, a no-operation verb has been designed which is to be used in conjunction with the IF ( ) THEN ( ) expression. It is called DCARE and is to be used as follows: IF ( ) THEN DCARE and serves to negate the positive branch of the IF. Flow would then always proceed

to the next instruction in sequence until the DCARE is replaced by another verb.

An area which has been neglected in authoring language design is that of recording and manipulating data for the author's use. This would involve recording particular responses and retrieving this information for analysis. The technique of counter use has been employed to accomplish this function. Usually the counters stand for a purpose, such as 'number of correct replies' or 'number of mismatches' etc.,. The counters are then periodically checked and the course flow altered depending upon their content. Writeacourse, Coursewriter II, and Vault are some of the well known languages that employ this technique.

One possible approach to the problem is to use counters, but identify each counter with the 'name' of each response of interest. Thus, a binary setting of the counter would indicate the presence or absence of the occurrence of the response.

Another area of neglect is that of path traversing. A path can be defined as the particular route a student took through the course. This information can be of special value to authors who individualize the current path through the course as a function of the previous execution of certain course BLOCKS. This has been provided through the automatic recording, by the authoring system, of each BLOCK a student traverses. This information can then be checked through the use of the IF PATH ( ) THEN ( ) verb.

Scorekeeping instructions generally involve the setting, incrementing, and interrogation of counters. The verb SET has been

designed to both define a counter and initially set it to zero. ADD and SUB have been adopted from various programming languages to increment and decrement counters. The conditional expression IF COUNT ( ) THEN ( ) has been designed to check the counter contents.

These counters can also be used to record the occurrence of a particular response. The counter name could be SET to the exact response spelling or a shortened version of it and automatically initialized to zero by the SET verb. Upon occurrence of the response, an ADD could be used to record the event. The setting of the counter can then be used to sequence the course.

Although a terminology, based on a subset of the natural English language, has been established for the langauge functions, it must be stressed that it is not fixed. New functions can also be added to the language. As the pre-processor is table driven, various terminologies can be used for the same functions by merely changing the verb name in the table. Thus, a greater flexibility is gained as the choice of terminology frequently depends upon the geographical location or the natural spoken language of the author.

3.3 Language Description

Each verb consists of two parts, the verb name itself and the associated parameter(s). The purpose and construction of each verb is now described within it's functional category.

1. Display information

   (a) TEXT — This verb is used to present information to the student on the systems output device. The parameter associated with

the verb is the actual text to be entered. Up to 150 characters may be entered.

2. Display information - request a response

(a) ASK — This verb is similar to TEXT in that information is presented on the systems output device, but differs in that the the information is a question and response request is initiated. The parameter associated with this verb consists of a question which may be up to 150 characters in length.

(b) CHOICE — This verb presents information on the systems output device. The choices are numbered for identification purposes. Response request is also initiated. The parameter consists of the choice to be entered. Up to 150 characters may be used.

3. Course sequencing

(a) DELAY — This verb is used to sequence the presentation of subject-matter through the use of hardware timing delays. A numeric quantity indicating the time delay in seconds is associated with this verb.

(b) GO TO — This verb causes unconditional transfer of control to either another BLOCK or to the previous BLOCK executed. There are two

possible parameters. An asterisk (*) indicates that control is to be transferred to the previous BLOCK executed. The other parameter is the label of a BLOCK to which control can be transferred.

4. Arithmetic

(a) SET — This verb defines and initializes a counter to zero. A counter name is associated with the verb.

(b) ADD — This verb increments a counter by a variable quantity. A counter name and a numeric quantity are associated with the verb.

(c) SUB — This verb is used to decrement a counter by a variable quantity. A counter name and a numeric quantity are the verb parameters.

5. Answer processing

(a) IF reply,reply,... THEN verb,verb,...

This verb is used to compare the anticipated reply (ies) to the actual reply (ies). The list of verbs following the THEN are executed if an exact match occurs on all of the anticipated replies listed in the expression. In all the answer processing verbs, the next instruction in

sequence is executed if an exact match does not occur. Also, in all conditional expressions, the verbs following the THEN may be chosen from a subset of the symbolic language.

(b) IF ANY reply,reply,... THEN verb,verb,...

This verb provides a means of specifying several acceptable replies. The anticipated reply (ies) is (are) compared to the actual reply (ies). The list of verbs following the THEN are executed if an exact match occurs on any of the anticipated replies listed in the expression.

(c) IF KEYWORD reply,reply,... THEN verb,verb,...

This verb can be used to identify the occurrence of keywords in the actual reply. The anticipated reply is checked in left to right sequence with the actual reply. A match is recognized if all the anticipated keywords are present in the reply.

(d) IF CHOICE number,number,... THEN verb,verb,...

This verb can be used to identify the selection of a choice or list of choices by the student. One or more numbers identifying the choice may be listed. The anticipated list of choice numbers is matched against the numbers of the actual

choices selected. A match is recognized upon a one-for-one match of both lists. The verbs following the THEN are executed upon a match.

6. Performance analysis

(a) IF PATH label,label,... THEN verb,verb,...

Each time a BLOCK is traversed by the student, the label of the BLOCK is recorded by the system. This sequence of BLOCK labels can be checked by this expression. The list specified in the expression is checked in left to right sequence against the recorded systems information. A match is recognized if the anticipated sequence of labels is found in the recorded information and will cause the verbs following the THEN to be executed.

(b) IF COUNT name,numeric quantity THEN verb,verb,...

This verb is used to check the contents of the counter named to the numeric quantity specified. A match will occur if the numeric quantity is greater than or equal to the counter contents. The verbs following the THEN are executed upon the satisfaction of the greater than or equal to condition.

3.4 Language Use

3.4.1 Introduction

The language verbs are to be used only within a BLOCK, which, starts with the verb BEGIN and terminates with the verb END. These two verbs, which structure the BLOCK, are not elements of the symbolic language, but belong to a class of verbs called systems verbs. This class of verbs will be described fully in Chapter four.

3.4.2 Restrictions

The author's teaching objectives are represented by sequences of verbs. Certain restrictions have been established as to how these sequences may be formed. They are as follows:

1. A verb which initiates response request must be immediately followed by at least one answer processing verb.
2. Answer processing verbs may only follow a verb which requests a response. If more than one is used, they must occur in an unbroken sequence.
3. An unconditional transfer verb may not be followed by any other language verb.
4. Answer processing verbs, performance analysis verbs, and verbs which request a response may not follow a THEN in an IF ( ) THEN ( ) type of verb.

These restrictions have been imposed upon the formation of verb sequences for two reasons. First, the pre-processor not only represents the verb by a data structure, but also guides the author in course formation. It maintains semantic control on the verbs entered

to ensure a proper and meaningful representation of the course. Therefore, the author is restricted, but in a sense of being guided rather than inhibited. The first three restrictions have been imposed for semantic control.

Secondly, it was considered desirable to impose limitations upon the use of the conditional type of expression IF ( ) THEN ( ). This restriction is the last one of the group stated. It was decided to prohibit the use of any conditional expression or a verb which requests a response following a THEN, as a conditional expression would then have to follow it - restriction 1. This restriction has been imposed primarily because the pre-processor interrogates the author for each parameter associated with the entered verb. Multiple use, or nesting of the conditional expression IF ( ) THEN ( ) would only complicate and confuse authoring and would not contribute any degree of power in analysis. A sequence of IF ( ) THEN ( ) expressions could accomplish the same purpose as a nested IF. The subset of allowable verbs following a THEN would then exclude all verbs that are affected by restriction 4. The language verbs which are not affected by the restrictions may be used in any sequence.

The semantic correctness of a sequence of verbs may be checked prior to entry by using the systems verb CHECK. The sequence of verbs associated with this verb will be checked and the author notified of any incorrect usage of verbs. This serves the purpose of allowing an author to check the validity of the sequence before he spends time entering information which only later is unacceptable because of an incorrect verb sequence.

Again it is to be noted that the restrictions, noted in the discussion, with the exception of that on the conditional expression, are not fixed and can be changed by modifying a master table of verbs and associated parameters. The first three restrictions are for semantic control while the last one is for systems control. The actual details concerning the effect of restrictions on the data structure will be discussed in Chapter four.

3.4.3 Verb Entry

Authoring consists of the author entering the verbs chosen to form the course in an interactive and interrogative environment. There are three stages involved in entering a verb. The author initiates the verb entry by typing in the name of the verb at the input terminal. The system then responds to this action by interrogating the author for each parameter associated with the verb. The author in turn replies with the requested information. These three stages will be referred to as author initiated (AI), system request (SR), and author response (AR). This cycle is performed for each verb.

Examples have been chosen from a medical course used at the University of Alberta and will be presented in three parts. The objective to be achieved is presented first, followed by the verb sequence used to accomplish this objective. The actual interactive and interrogative entry of these verbs will then be discussed in terms of the three stages. The first stage will be referred to as AI (author initiated). The second and third stages are related and will be identified by SR-AR (system request - author response). Occasionally, the system may just respond to the author without actually requesting any information.

This situation will be identified by an (S) for system. Finally, the information presented by the author is in lower case letters while systems responses can be identified by the use of upper case letters.

EXAMPLE 1

Objective: To present the following text with the indicated timing delays:

"Palpitation occurs concurrently in patients with heart disease and may signify:

1. Increased ventricular stroke volume (as in high cardiac output states, valvular imcompetence, intracardiac shunts, etc.,)"

(delay next presentation 10 seconds)

2. "Arrhythmias such as extrasystoles, paroxysural vachycardia, atrial fibillation, etc.,"

(delay next presentation 10 seconds)

3. "Concern over cardiac malfunction with heightened awareness of heart action."

Verb sequence: BEGIN,TEXT,TEXT,DELAY,TEXT,DELAY,TEXT,END

Verb entry:

```
AI      begin
SR-AR   LABEL=6.02b

AI      text
SR-AR   TEXT=palpitation occurs concurrently in patients with
             heart disease and may signify:

AI      text
SR-AR   TEXT=1. increased ventricular stroke volume (as in high
             cardiac output states, valvular incompetence,
             intracardiac shunts, etc.,)

AI      delay
SR-AR   SECONDS=10
```

```
AI      text
SR-AR   TEXT=2. arrhythmias such as extrasystoles, paroxysural
             vachycardia, atrial fibillation, etc.,

AI      delay
SR-AR   SECONDS=10

AI      text
SR-AR   TEXT=3. concern over cardiac malfunction with heightened
             awareness of heart action.

AI      end

S       LABEL 6.02b ENDED
```

EXAMPLE 2

Objective: To present information, ask a question and process a single reply as follows:

"The important points are fatigue, palpitation, cardiac enlargement recognized from early childhood, and possible chest pain."

"In a patient with heart disease, what is the significance of fatigue?"

If the reply is 'poor tissue perfusion', add 2 to the score and branch to BLOCK 6.01b, otherwise, branch to BLOCK 6.02.

Verb Sequence: BEGIN,TEXT,IF ( ) THEN ADD,GOTO;, GOTO, END

Verb entry:

```
AI      begin
SR-AR   LABEL=6.01

AI      text
SR-AR   TEXT=the important points are fatigue, palpitation, cardiac
             enlargement recognized from early childhood, and possible
             chest pain.

AI      ask
SR-AR   QUESTION=in a patient with heart disease, what is the
                 significance of fatigue?

AI      if
SR-AR   IF: poor tissue perfusion
SR-AR   THEN: set,add,goto
```

```
S       VERBS OK
                                  (The parameters for each verb following
                                   the 'THEN:' will now be requested by
                                   the system.)
SR-AR   COUNTER=cnt1              (for SET)
SR-AR   COUNTER=cnt1              (for ADD)
SR-AR   HOW MUCH ? 2              (for ADD)
SR-AR   LABEL=6.01a               (for GO TO)

                                  (The system now interrogatively directs
                                  the author to entry of the next verb
                                  in sequence.)
S       AND IF NOT ?

AI      goto
SR-AR   LABEL=6.02

AI      end

S       LABEL 6.01 ENDED
```

EXAMPLE 3

Objective: This example illustrates the use of some of the conditional answer processing verbs that can be used for complex analysis. It will represent the following situation:

"What were the significant points in the history?"

(a) If reply is 'fatigue,palpitation, large heart (age 6)', and 'chest pain', add 3 to score and branch to BLOCK 6.00a.

(b) If reply is only 'chest pain', then branch to BLOCK 6.00d.

(c) If reply consists of any of the following: 'fatigue, palpitation, large heart (age 6)', then add 1 to score and branch to BLOCK 6.00c.

(d) If none of the above conditions occur, delay for 30 seconds

and branch to BLOCK 6.00e.

Verb sequence: BEGIN,SET,ASK, IF ( ) THEN ADD,GOTO;, DELAY,GOTO, END

Verb entry:

```
AI      begin
SR-AR   LABEL=6.00

AI      set
SR-AR   COUNTER=cnt1

AI      ask
SR-AR   QUESTION= what were the significant points in the history?

AI      if
SR-AR   IF: fatigue,palpitations, large heart (age 6)
SR-AR   THEN: add,goto

S       VERBS OK

SR-AR   COUNTER=cnt1            (for ADD)
SR-AR   HOW MUCH ? 3            (for ADD)
SR-AR   LABEL=6.00a             (for GO TO)

S       AND IF NOT ?

AI      if
SR-AR   IF: chest pain
SR-AR   THEN: goto

S       VERBS OK

SR-AR   LABEL=6.00d

S       AND IF NOT ?

AI      if any
SR-AR   IF: fatigue, palpitation, large heart (age 6)
SR-AR   THEN: add,goto

S       VERBS OK

SR-AR   COUNTER=cnt1            (for ADD)
SR-AR   HOW MUCH ? 1            (for ADD)
SR-AR   LABEL=6.00c             (for GO TO)

S       AND IF NOT ?

AI      delay
SR-AR   SECONDS=30
```

```
AI      goto
SR-AR   LABEL=6.00e

AI      end

S       LABEL 6.00 ENDED
```

EXAMPLE 4

Objective: To ask a question, present several choices for selection, and identify choices selected as follows:

"Can you make a firm diagnosis?"

(a) If 'yes', then branch to BLOCK 5a, otherwise branch to BLOCK 5.3.

Verb sequence: BEGIN,ASK,CHOICE,CHOICE, IF CHOICE ( ) THEN GOTO;, GOTO, END

Verb entry:

```
AI      begin
SR-AR   LABEL=5a

AI      ask
SR-AR   QUESTION=can you make a firm diagnosis?

AI      choice
SR-AR   CHOICE 001 = yes

AI      choice
SR-AR   CHOICE 002 = no

AI      if choice
SR-AR   IF: 1
SR-AR   THEN: goto

S       VERBS OK

SR-AR   LABEL=5a

S       AND IF NOT ?

AI      goto
SR-AR   LABEL=5.3

AI      end

S       LABEL 5.00 ENDED
```

EXAMPLE 5

Objective: This example illustrates the use of performance analysis verbs to individualize a course based upon a particular path a student has traversed in the course. The conditions are as follows:

> "You must be deaf. Listen again - there is a loud pansystolic murmur."

(a) If BLOCK 5.1 followed 5.0 then branch to BLOCK 9.

(b) If BLOCK 5.2 followed 5.0 then branch to BLOCK 8.

(c) If neither condition occurs, then proceed sequentially to the next BLOCK.

Verb sequence: BEGIN,TEXT,IF PATH ( ) THEN GOTO;, IF PATH ( ) THEN GOTO;, END

Verb entry:

```
AI      begin
SR-AR   LABEL=Q8b

AI      text
SR-AR   TEXT=you must be deaf.  listen again - there is a loud
             pansystolic murmur.

AI      if path
SR-AR   IF: 5.0,5.1
SR-AR   THEN:goto

S       VERBS OK

SR-AR   LABEL=9

S       AND IF NOT ?

AI      if path
SR-AR   IF: 5.0,5.2
SR-AR   THEN: goto

S       VERBS OK

SR-AR   LABEL=8

S       AND IF NOT ?
```

```
AI      end
S       LABEL Q8b ENDED
```

3.5 Patterns

Teaching strategies often require repetitive use of verb sequences. The strategy of Drill and Practice for example, consists of repetitive question asking and answer processing. As the entry of identical verb sequences is time consuming and tedious, the author may define certain verb sequences and also provide a name by which the sequence may be referred to (pattern). These patterns may be called by name and used as if they were verbs of the symbolic language. Upon entry of a pattern name, the parameters associated with each verb in the pattern will be requested of the author. For example, the pattern INQ exists, which consists of the following verb sequence: ASK, IF ( ) THEN ( ). Another pattern, called TUTOR, represents the commonly used tutorial teaching strategy which presents information, asks a question and processes the reply. It consists of the verb sequence TEXT,ASK, IF ( ) THEN ( ).

These two patterns have been included in the system as an example. However, patterns are not fixed as they are also a part of the master table that drives the pre-processor, and can be created by adding to this table. The author has this capability in the systems verb CREATE. It is to be used in the same interactive manner as symbolic language verbs. Upon entering the verb CREATE, the system would request the verb seqeunce and the name to be associated with the sequence. The name must be unique, that is, not the name of another pattern, symbolic language verb, or systems verb. Prior to establish-

ing this sequence as a pattern, several checks are performed by the system. The verb sequence entered must be semantically correct and not violate the language restrictions. Pattern names must not be among the verb sequence. The following example demonstrates the creation and use of a pattern.

EXAMPLE 6

Objective: To create and use a pattern that will display text, followed by a timing delay, and then branch to a BLOCK.

Verb sequence: CREATE,BEGIN,NEWPAT,END

Verb entry:

```
AI      create
SR-AR   NEW NAME= newpat
SR-AR   VERBS=text,delay,goto

S       VERBS OK

AI      begin
SR-AR   LABEL=usepat

AI      newpat
SR-AR   TEXT=your clinical diagnosis was correct.
SR-AR   SECONDS=3
SR-AR   LABEL=10.0

AI      end

S       LABEL usepat ENDED
```

The computer entry of EXAMPLES 3,4, and 6 are listed in Appendix A.

CHAPTER IV

LIST STRUCTURE REPRESENTATION

4.1 Course Modelling

The language and parameters described in Chapter three are the input data to the pre-processing system. The final objective of this system is to produce a model of the entered information, i.e., a representation of the course. A machine code program can then be produced from this representation by a multi-lingual translator.

Several factors affect the design of a course model. As authoring (verb entry) may be extended over a period of several authoring sessions, the model would be constructed in sections. Thus, the author must have access to all parts of the course model for modification purposes during authoring time. Therefore, it must be capable of growth within the context of its' formation, and also be precise in format as this is an input requirement of translators in general. Finally, the course model must be capable of handling large quantities of variable length data.

The development of list processing languages such as SNOBOL, LISP, and L6 have greatly facilitated the technique of modelling through the use of list structures. This modelling technique involves the use of linked lists in which each element to be represented is an entry in the list and is related to the previous list entry. Figure 4.1 illustrates the structure of two types of lists.

In a Linear list, each entry is connected to the following element. A Ring list is similar but also relates the last list entry Pointers within each list element are used to accomplish the linkage.

Sub-lists can also be attached to the main list by the use of pointers. Figure 4.2 illustrates a list structure using sub-lists.

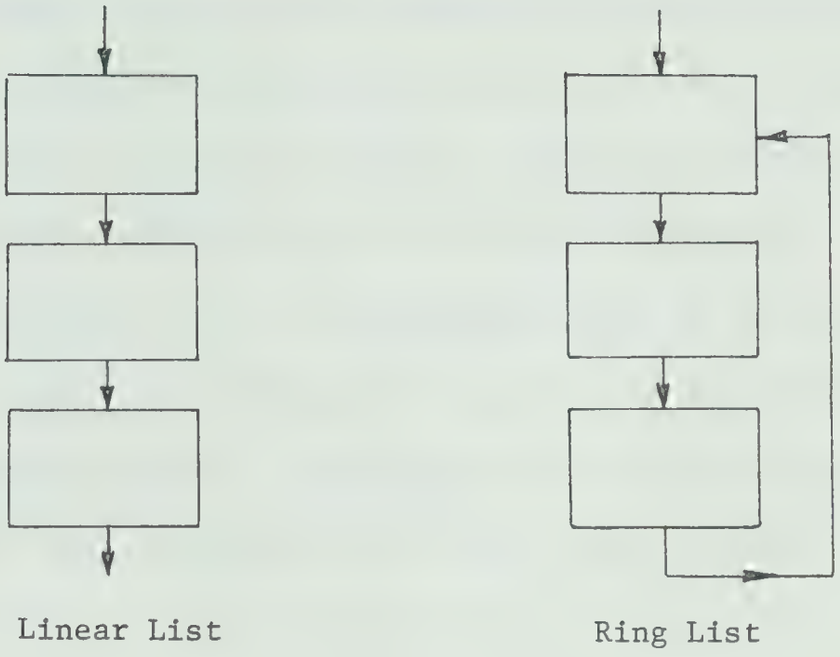

Figure 4.1 Two Types of Linked Lists

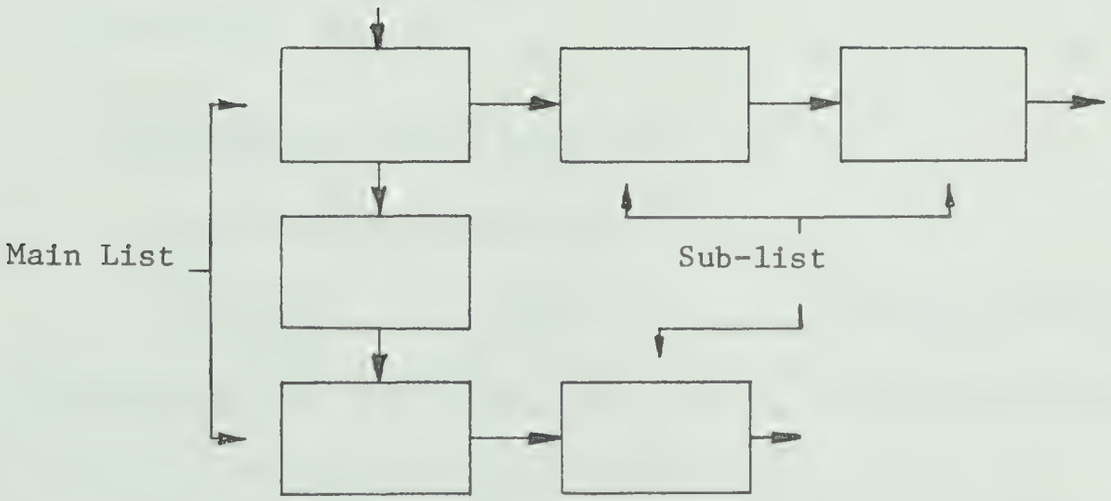

Figure 4.2 A Linked List With Sub-lists

Some work has been done in the use of list structures for modelling CAI courses. A list structural representation of Coursewriter II (CW II) instructions was developed by Flathman (Flathman 1969). The primary concern of this work was to translate CW II instructions into a list structural model of the course. From this model, a flowchart of the instructional logic was produced, indicating the logical paths a student can follow through the course.

The characteristics of a structure-system to be modelled often determine the modelling procedures. A model of a CAI course, as utilized in this system was considered to have properties delimiting the modelling to that best represented by a list structure. Therefore, the design of the model has been based upon this type of modelling technique.

Section 4.2 discusses the establishment of a model representation. The formation of a list structure is described in section 4.3. Section 4.4 then describes the content of each verb representation in the structure.

4.2 Establishing a Model Representation

4.2.1 A Model Structure

A course is, in a technical sense, a computer program. Therefore, properties of programs and those of programming languages directly influence the design of the program model.

A program can be considered as a sequence of programming language instructions. The relation of each instruction to the other creates an organization pattern. This pattern organizes the instructions.

A representation of the instruction organization (structure) of a program must be established that can accurately represent most typical CAI programs. This type of program is often characterized by two types of structures used to organize the programming language instructions. These are illustrated in Figure 4.3.

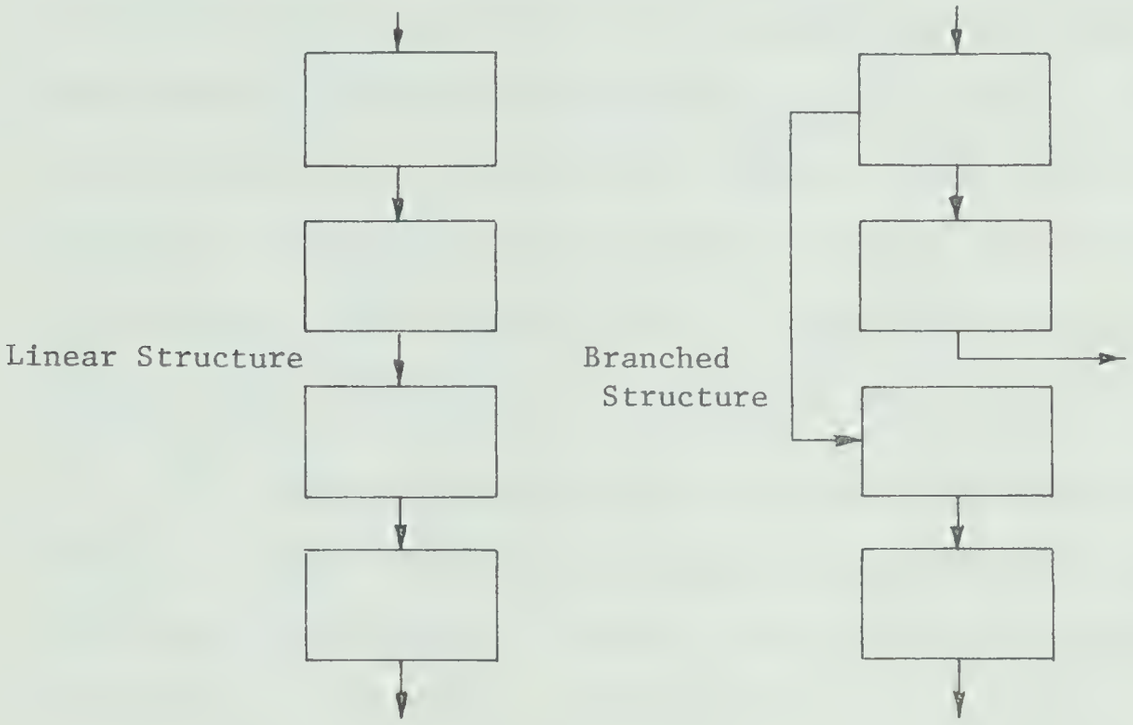

Figure 4.3

Generally, the overall physical structuring of the instructions of a CAI program can be represented by the integrative use of both types of structures illustrated in Figure 4.3 and also the use of lists with attached sub-lists as illustrated in Figure 4.2 (George 1966).

The identifying feature of this type of instruction structure is the use of groups of verbs. The physical locational relationship between any two groups is basically one of the following two types:

1. The group is connected to the next group in sequence, or

2. it is connected to another group in the program.

As this type of organization can represent most CAI coding techniques, it has been used as the super-structure or skeleton within which the symbolic verbs are to be modelled. A group is identified as a sequence of verbs of which only the first is labeled. Thus nesting of groups has not been allowed. The systems verbs BEGIN and END have been created to form the beginning and end of the group. BEGIN is the only verb that may be labeled in the course. These verbs have been assigned list structural representations and are part of the resulting course model. Thus, the structuring or organization of a course is entirely under the author's control through the formation of groups.

The relationship between groups is also determined by the author. The author enters the symbolic verbs within the framework of a BEGIN and END structure. This was illustrated in the examples presented in section 3.4.3. If BEGIN-END groups are defined sequentially, that is, upon entering an END, the author then starts a new group with a BEGIN, the two groups are automatically linked together. This results in a linear grouping. The author may wish to alter this relationship and break the link, as may be desired if the group just ended is to be linked with another group in the course. A branched group can be achieved by using the systems verb TERMINATE. Figure 4.4 illustrates the resulting structure.

Therefore, each group may be developed individually and later connected. The resulting structure is then in a straightforward modular form that is most suitable for efficient code generation.

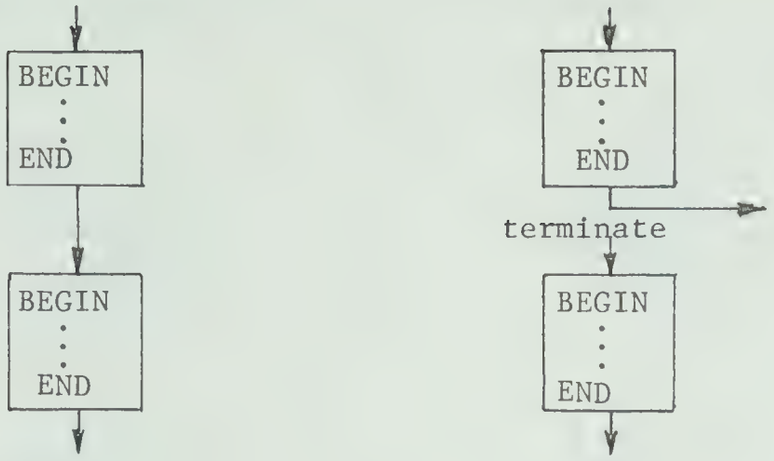

Figure 4.4 Two Types of Group Linkage

4.2.2 Symbolic Verb Modelling

The symbolic verbs have been modelled within the group structure established in section 4.2.1. As authoring languages are a subset of computer programming languages, the properties of programming language elements in general have been considered in the establishment of a list structural representation of the symbolic language verbs.

A program consists of a complex union of many instructions. These instructions are joined to form a hierarchy of commands to the computer. The common properties of verbs could be established in terms of verb function, language form, or a variety of other detailed language aspects. However, these factors are not standard for the wide range of existing languages. One feature of language verbs, however, is standard for all programming languages. It can be described as the effect a verb has on the path flow of a program and will be termed logical path flow.

Verbs can be classified into two types of flow categories. A linear type verb allows the logical path to proceed linearly to the next verb in sequence. A branched or conditional type verb alters the path flow or has the capability of altering it. Figures 4.5 and 4.6

illustrate the representation of each flow category.

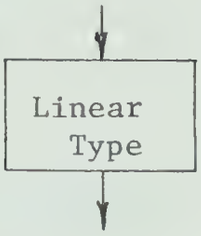

Figure 4.5

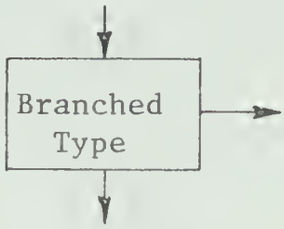

Figure 4.6

This categorization according to type of linear flow has been used as a basis of the list structural representation of the symbolic language verbs. A linear type verb will be referred to as Type 1 and a branched verb as Type 2 in further discussions.

The symbolic language verbs presented in Chapter three have been categorized according to Type and assigned identification codes. These are listed in Table 1.

Each 'XXXXXXXXXXXX' entry in the table is used to represent and eventually construct a 'dummy' vector that represents the negative or 'and if not' branch of the preceding IF in the list. The use of this 'dummy' vector will be described later.

The verb type has been pre-assigned based upon the logical path flow of the verb, and is recorded along with other verb information in a master table of verbs.

The verbs have been further classified based on a variable parameter that is set after the verb is entered by the author. This parameter indicates the presence of data associated with the verb. There are two possible conditions. Simply stated, a verb either has

VERB	CODE	TYPE
TEXT	01	01
ASK	02	01
IF	03	02
XXXXXXXXXXXX	04	02
IF ANY	05	02
XXXXXXXXXXXX	06	02
IFANY	05	02
XXXXXXXXXXXX	06	02
IF KEYWORD	07	02
XXXXXXXXXXXX	08	02
CHOICE	09	01
IF PATH	10	02
XXXXXXXXXXXX	11	02
IF COUNT	12	02
XXXXXXXXXXXX	13	02
SET	14	01
ADD	15	01
GO TO	16	01
DELAY	17	01
SUB	18	01
DCARE	19	01
IF CHOICE	20	02
XXXXXXXXXXXX	21	02

Table 1

Codes and Types of Symbolic Verbs

associated parameters (data) or it does not. Four types of verb representation exist depending upon combinations of logical path flow and parameter specifications. Figure 4.7 illustrates the four cases.

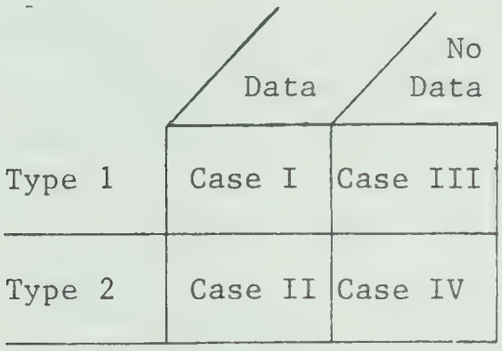

Figure 4.7 Verb Classification Matrix

Each symbolic verb has been assigned a representation corresponding to one of these cases. Systems verbs that are part of the data structure, such as BEGIN and END have also been assigned a representation based on this type of categorization. Each case has a unique representation in the data structure. This representation becomes the basis for defining a cell in the list structure.

4.3 Structure Formation

4.3.1 Cell Linkage

The symbolic language verbs are used only within a BEGIN-END group. These two systems verbs have been assigned a list representation and are part of the structure. Therefore, the entire group of verbs is represented in the model. By convention, let Figure 4.5 (see section 4.2.2) represent a Type 1 verb and Figure 4.6 (see section 4.2.2) represent a verb of Type 2. Figure 4.8 illustrates a list model of the

following verb sequence: BEGIN, ASK, IF ( ) THEN DELAY GOTO;, END. The entry sequence number is given in each box.

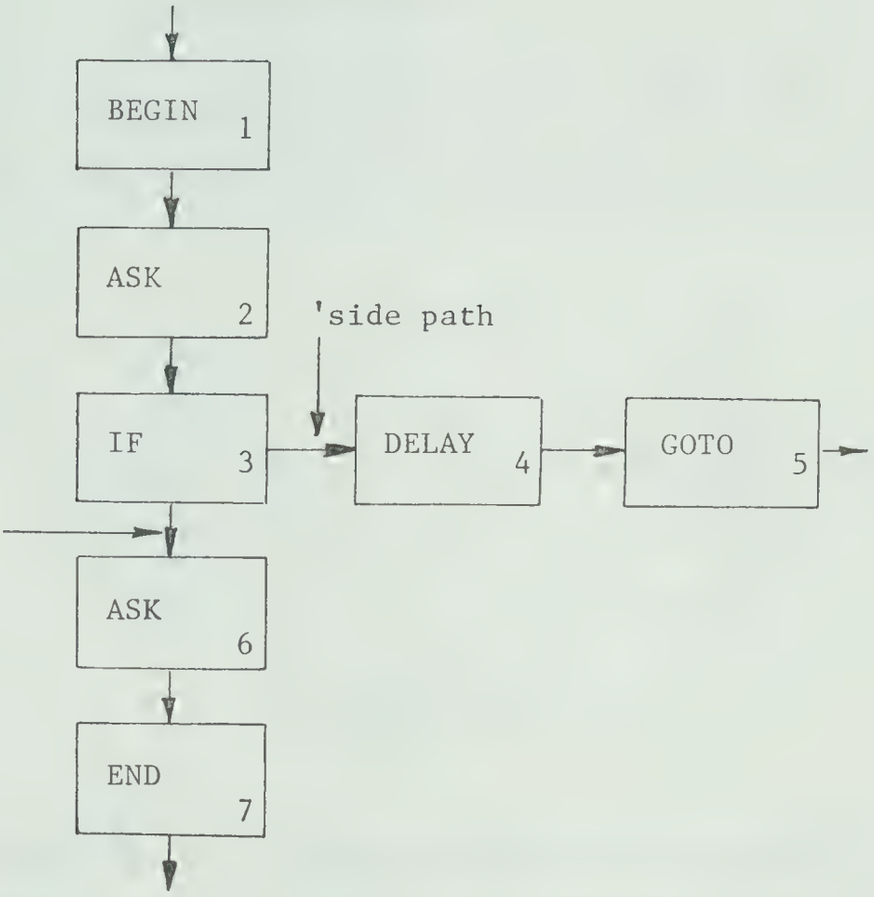

Figure 4.8 List Model of a Verb Sequence

As each verb is entered, it is checked for semantic compatability to the last verb in the structure. If compatability is established, the new verb is linked. Linkage depends upon the verb Type. Type 1 verbs are linked linearly while Type 2 verbs first initiate the linkage of a 'side path'. This can be described as the path which contains the cell representations of the verbs following a 'THEN:'. In Figure 4.8, the verbs DELAY and GO TO constitute the 'side path'. After each verb following a THEN is entered into the structure, the

verb code of the IF is changed to that of the associated 'dummy' verb. In this manner indication is given to the system that flow is to resume linearly.

4.3.2 Linkage Restrictions

Each Type 1 cell has an input and output node. Type 2 cells have three nodes, one for input and two for output. This is shown in Figure 4.9.

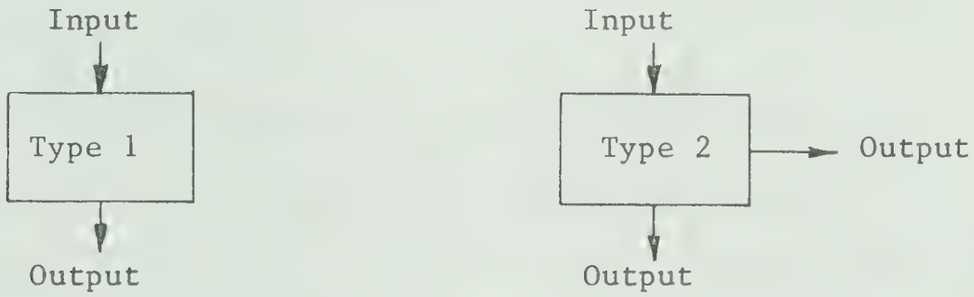

Figure 4.9

Often illogical verb sequences are chosen and the use of these causes major logic errors in the course. In order to eliminate semantic inconsistencies and logic errors, the pre-processor has been designed to check the semantic compatability of each newly entered verb with the verb previously entered. This is done by restricting the input-output node connections of each verb. Imposing such restrictions implies that each verb in the language can only be linked to a verb with which it is semantically compatible. A semantically incompatible verb set would be DELAY followed by IF CHOICE ( ) THEN ( ), as the conditional expression logically should follow the presentation of a choice (CHOICE). Another more familiar type of error is a GO TO followed by TEXT.

Selected codes from those assigned to each symbolic verb (see Table 1) have been used to form a set of restrictions for each verb. This set represents the identification codes of verbs to which the verb being restricted cannot be linked. The restrictions associated with each verb are listed in Table 2. The restriction set also includes codes assigned to systems verbs. These codes begin with an 'S' and will be described in Chapter five. The entire set of restrictions are part of a master file of verbs and associated information, and may be modified by changing the file.

The pre-processor compares the identification code of the last verb represented in the structure to the list of restrictions associated with the next verb to be represented. If the identification code is among those in the restriction set, a message is returned to the author indicating the error.

The 'dummy' code associated with each Type 2 verb is also used in the check for semantic compatability. As any Type 2 verb has two output nodes, each node can be linked to an input node of another verb. However, the 'side path' of a Type 2 can only be linked to certain verb sequences, while the 'and if not' path of a verb cell can possibly link to other verbs in the language. Therefore, each Type 2 verb is assigned two identification codes. The code associated with the verb in Table 1 is the code of a Type 2 verb as related to the 'side path'. The following 'dummy' code represented by the verb 'XXXXXXXXXXX' represents the 'and if not' path. The dummy code does not have restrictions of its' own but may be listed in the restriction set of other verbs. Syntactical checks on a Type 2 verb are conducted as follows: The verb code associated with the Type 2 verb in Table 1

CODE	VERB	RESTRICTION SET
01	TEXT	02091619SASCSESFSGSISJ
02	ASK	020305070910121619 20SASCSESFSGSISJ
03	IF	0103050709101213141516171819202 1SASBSCSE SFSGSISJ
04	XXXXXXXXXXXX	--
05	IF ANY	0103050709101112131415161718192021SASBSC SESFSGSISJ
06	XXXXXXXXXXXX	--
07	IF KEYWORD	0103050709101112131415161718192021SASBSC SESFSGSISJ
08	XXXXXXXXXXXX	--
09	CHOICE	030507101216192 0SASCSESFSGSISK
10	IF PATH	020305070910121619 20SASCSESFSGSISJ
11	XXXXXXXXXXXX	--
12	IF COUNT	020305070910121619 20SASCSESFSGSISJ
13	XXXXXXXXXXXX	--
14	SET	02091619SASCSESFSGSISJ
15	ADD	02091619SASCSESFSGSISJ
16	GO TO	02091619SASBSCSESFSGSISJ
17	DELAY	020305070917 19SASCSESFSGSISJ
18	SUB	02091619SASCSESFSGSISJ
19	DCARE	0102040608091113141516171819 21SASBSCSDSE SFSGSHSISJ
20	IF CHOICE	0102030405060708091011121314151617181920 SASBSCSDSESFSGSHSISJ
21	XXXXXXXXXXXX	--

Table 2

Symbolic Verb Restrictions

is used to compare with the code of the first verb in the 'side path'. The dummy verb code is assigned to the Type 2 verb upon completion of the 'side path' compatibility check. The next linear entry is then checked against the 'dummy' Type 2 code.

Verb sequences can be pre-checked by the use of the systems verb CHECK. The restrictions of each verb in the sequence are quickly analyzed and the author immediately informed as to the semantic compatability of the verbs prior to entering them. This type of pre-checking is performed on all verbs following a THEN by the system prior to construction of cell representations along the 'side path'.

The concept of restriction, as employed in this system, is simple and yet highly flexible as codes can readily be modified. A high degree of control is also established thus ensuring semantic and syntactic soundness in the resuting data structure.

4.4 Cell Content

The contents of a cell depend upon the needs of the model. In the simulation of CW II instructions, Flathman represented each instruction by a 10 byte cell. The data associated with each CW II instruction, such as an instruction label, was also represented by a cell. The cells containing such data were linked to the cell containing the instruction. Therefore, all information related to and including the verb was represented within the cell structure as opposed to storing the data elsewhere. In Flathman's structure, a cell contained four types of information as follows:

1. A 2 byte integer field for identification,

2. A 2 byte integer field for the pointer to the previous cell in the list,
3. A 2 byte integer field pointer to the next cell in the list, and
4. A 4 byte field for the contents of the cell (data).

Parameters and textual material associated with each CW II instruction were not represented as they did not contribute to the analysis of path flow.

In the model under design, each symbolic verb has been represented by a cell in the course model. This cell contains information about the verb and also information necessary to model the verb. In the course model established for the symbolic language, certain criterion were used to determine the exact contents of a cell. First of all, the identification of each verb must be included in the cell. Codes were assigned to each verb for this purpose as opposed to using the verb name (see Table 1). The verb Type was also stored as it is used to identify the cell Type during cell construction.

The use of a list structure in modelling the course immediately indicates the need for pointers within the cell. Pointers have been established to connect each cell to the preceding and following cell in the structure and have been included within the cell.

List structures often do not require two pointers. A pointer to the following cell is sufficient if the structure is only traversed in a one-way downward path. However, in the model being designed, two-way pointers were found to be necessary as the resulting structure would be traversed in both directions. An additional pointer is

necessary for Type 2 verbs as a 'side path' linkage is necessary.

In order to identify verbs within a group, a unique cell identification code must be assigned to each verb within a group. This is necessary for editing or retrieving cells from the structure. The first cell in a group is assigned a numeric identification code starting with '01'. Further cells are assigned numerically increasing codes. The cell can then be retrieved by using the BLOCK label and the cell id code.

Most of the symbolic verbs have associated data or parameters. There are two possible techniques that can be used in modelling this information. First, the information can be stored in cells also. This technique was used by Flathman and is advantageous if small quantities of data are associated with the verb. The second approach is to maintain a pointer to the data in the cell and store the data in a table or external storage device. As the information associated with each verb is variable in length and large in quantity, the latter approach would be more appropriate. Therefore, a pointer to the associated information has also been included within the cell.

Finally, it is always wise to allow extra space within a representation for further expansion. Therefore, a small part of the cell was left unused.

All or part of the cell parameters may be present within a given cell. The four cases stated in Figure 4.7 were used to determine the information maintained in the cell. The following information has been included in all four types of cells (Case I - Case IV):

1. Verb id - 2 byte integer field
2. Verb Type - 2 byte integer field
3. Pointer up - 4 byte hexadecimal address
4. Pointer down - 4 byte hexadecimal address
5. Cell id - 2 byte integer field
6. Number of associated data bytes - 2 byte integer field

If a verb has associated data, the cell representing it contains the following field (Case I - Case II):

7. Data pointer - 4 byte numeric number

The data associated with a verb is stored in a file. This field contains the address of the data in the file.

Finally, conditional or Type 2 verbs (Case II,IV) have the following field in addition to all the others mentioned:

8. Pointer to 'side path' - 4 byte hexadecimal address

In addition to the above fields, an extra 4 bytes has been included in all cases for expansion.

The actual cell structure of each case can be observed the following figures.

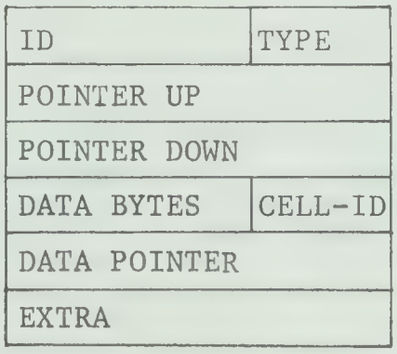

Figure 4.10 Type 1 - verb + data

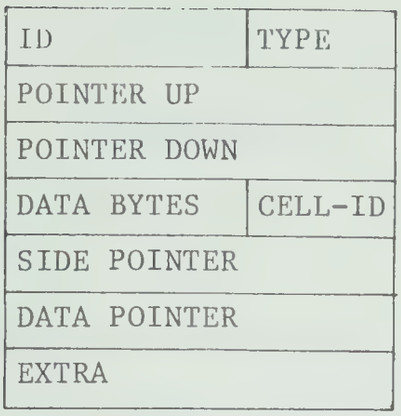

Figure 4.11 Type 2 - verb + data

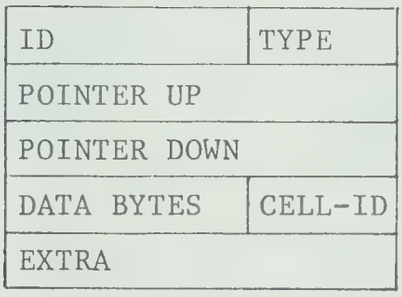

Figure 4.12 Type 1 - verb + no data

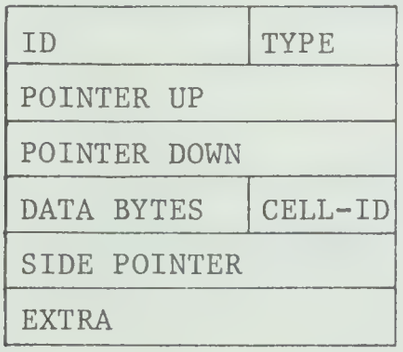

Figure 4.13 Type 2 - verb + no data

The cell size varies from 5 to 7 words. The actual size is not determined until a verb is entered and its' type and data parameter checked. The appropriate number of bytes is then assigned to the cell.

CHAPTER V

SYSTEMS DESIGN

5.1 Overall System View

The pre-processing system proposed in section 2.4.1 is designed to be part of a larger, three stage authoring scheme as illustrated in Figures 5.1, 5.2, and 5.3.

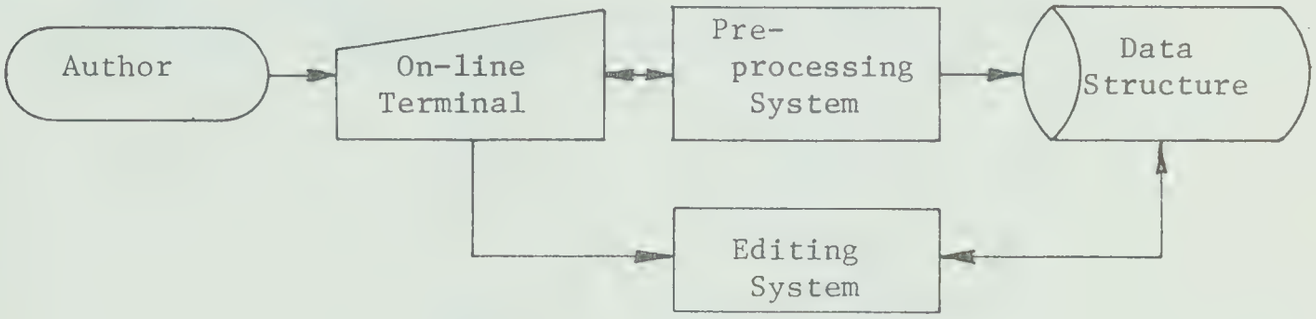

Figure 5.1 Stage 1 - Structural Representation

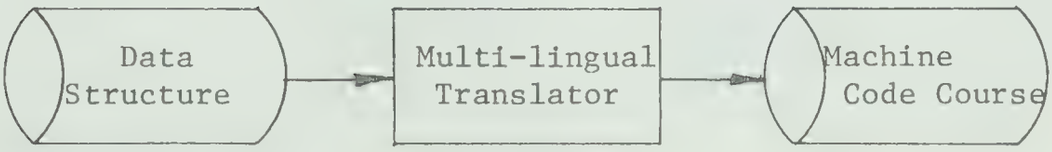

Figure 5.2 Stage 2 - Code Generation

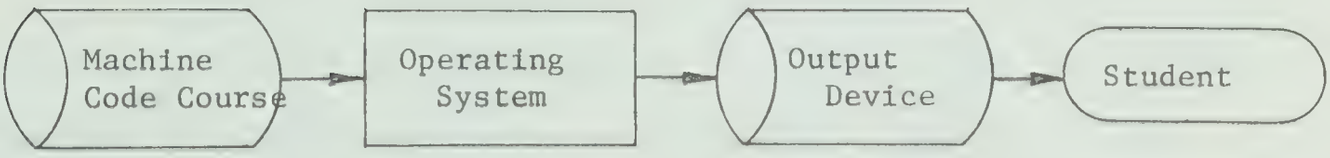

Figure 5.3 Stage 3 - Course Execution

Of this scheme, Stage 1, the pre-processing system has been developed in this thesis. The components of this system and their inter-relationships are illustrated in Figure 5.4.

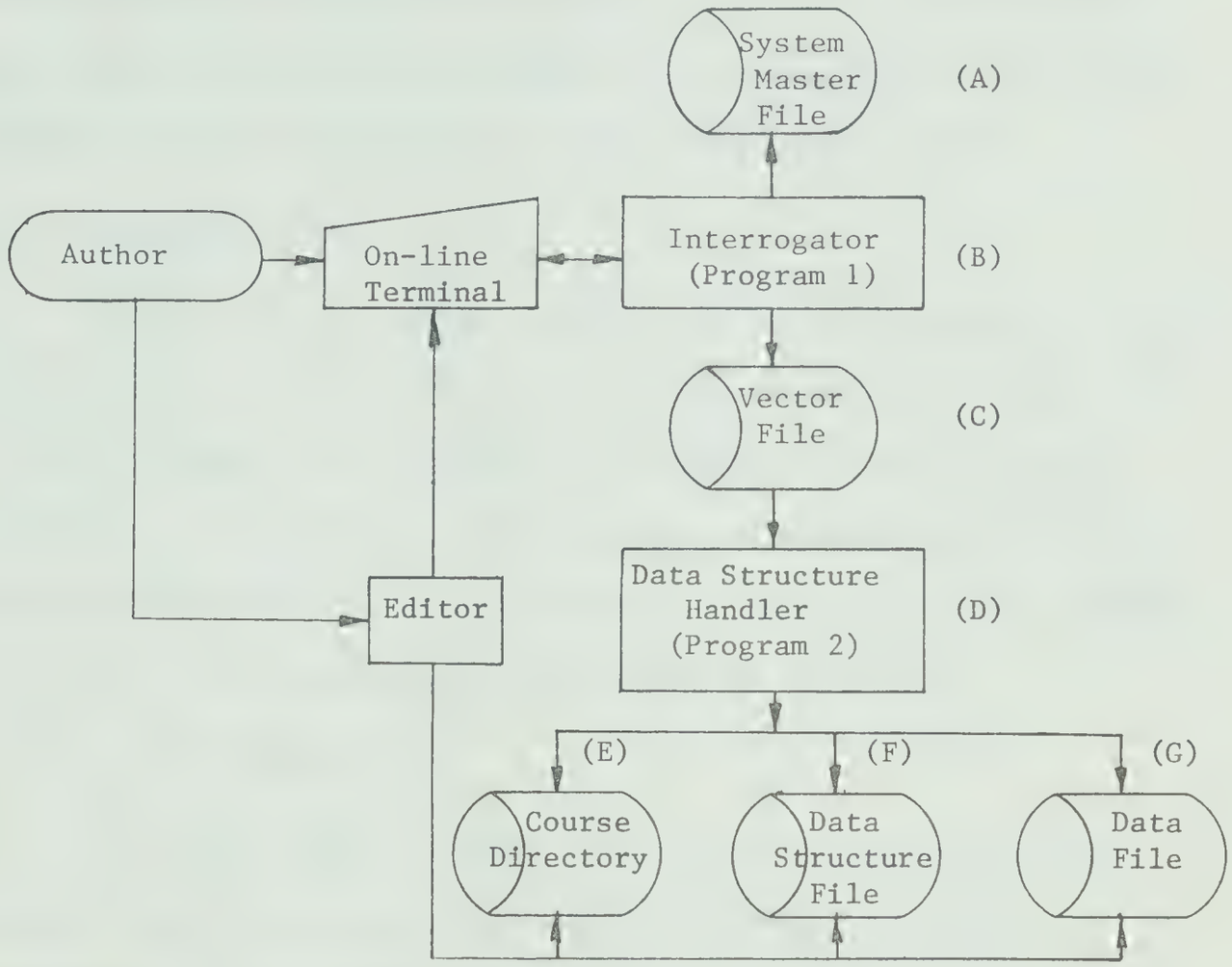

Figure 5.4 The Pre-processing System

This system consists of a symbolic language, two programs and several files used for systems control and data storage and has been designed for implementation on the IBM 360/67 computer under the Michigan Terminal System (MTS). The system illustrated in Figure 5.4 has been implemented with the exception of the Editor.

This chapter discusses the design of the components comprising the pre-processing system and their use in constructing a course model. Section 5.2 discusses the flow of information and the general purpose of the files illustrated in Figure 5.4. A detailed format description of these files is presented in section 5.3. Section 5.4 presents a description of the systems commands that are used to control course construction and to access the resulting data structure.

5.2 Information Flow

When an author enters a verb at an on-line terminal, a network of information flow is initiated. In order to transform a verb into a list structure cell, a series of information files are used and maintained. Some of these files contain permanent systems information while others store the data structure and associated control information. The flow of information will be described in two stages.

5.2.1 Stage 1

The first stage consists of author interrogation and the construction of an input vector to the second stage. Figure 5.5 illustrates the components of Stage 1.

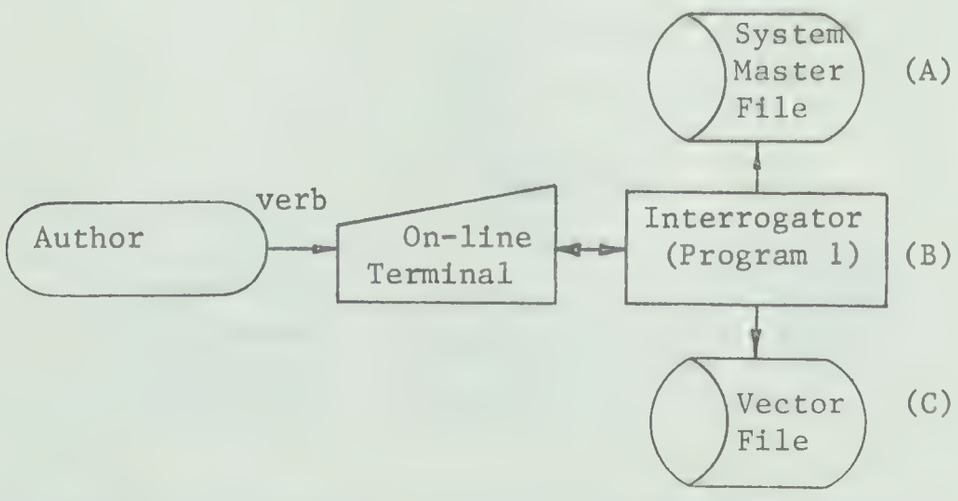

Figure 5.5 Stage 1 - The Pre-processing System

The first program (B) of the pre-processing system is responsible for the conversational systems interaction with the author. After a verb is entered, the system master file is accessed to identify the verb and access the control information associated with it. Upon identification of a verb, the program interrogates the author for related parameters (if any). A vector is constructed containing the entered verb, its' parameters and the system control information obtained from the Master file. This vector is stored in the Vector file. The information contained in the stored vector is used by Stage 2 to create a list structure representation of the verb. Stage 1 terminates with a call of the second program (D). This initiates Stage 2.

5.2.2 Stage 2

The second stage involves the use of the vector generated by Stage 1 to create a data structure representation of the verb. Figure 5.6 illustrates the components of Stage 2.

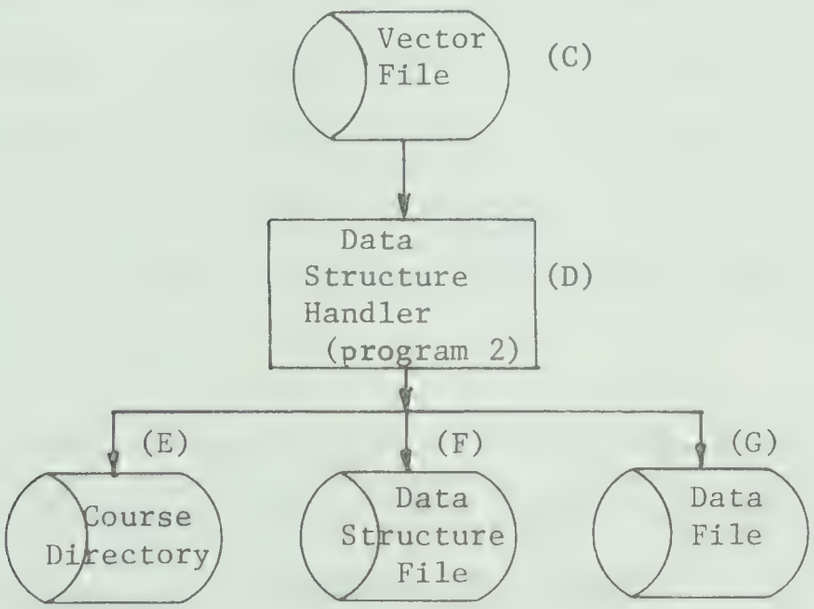

Figure 5.6 Stage 2 - The Pre-processing System

Stage 2 begins when the second program (D) is called by the program in Stage 1 (B). First, the vector is obtained from the Vector file. It may represent a symbolic language verb or a systems verb. The latter case will be discussed in Chapter four.

The first vector to be received by program 2 (D) is a sign-on vector and contains information identifying the course. The course directory (E) is then accessed to determine the status of a course. This file contains the course name and author identification. Information related to the data structure representation is also kept. This information is recorded as entered. If the course has been established during a prior authoring session, the information in the Course file is used to re-establish controls to allow authoring to continue from the last entered verb. Otherwise, a new course record is prepared.

After processing the sign-on vector, the program establishes controls to begin cell construction. Stage 2 is mainly responsible for transforming the vector representation of the verb into a cell representation. This representation is maintained in core. As each cell is constructed, the data associated with the verb (if any) is immediately stored on an output file (G). The address of this data is stored until a sign-off request is received. Upon this condition, the course directory (COURSE) is updated and the cell list stored on the Data Structure file (F).

After each construction of a cell representation, control is returned to the program in Stage 1 for further verb entry. This cycle is repeated for each verb entered by the author.

The specific details concerning each information file is now presented.

5.3 Systems Files

5.3.1 Introduction

The MTS System has special file properties which have been advantageous to the operation of the authoring system developed. There exist two types of files in MTS, line and sequential. Line files have been used for the storage of systems information and data. A line file is composed of a certain number of lines, depending upon the user's file space allocation. Each line can represent up to 256 bytes of information. The file may be accessed sequentially or by a specific line number. Another important characteristic is the unconverted representation of data. For example, one data record may contain a variety of data formats such as hexadecimal, character, or integer. This data is stored in a line in its' defined form. The systems handling of data record length is also of special interest. Unless otherwise specified, a data record is trimmed, that is, all trailing blanks except one, are dropped and the record then stored in the file. This dynamic file space allocation has been considered especially advantageous as many of the authoring system files contain variable length data records.

5.3.2 Control Files

Control files contain information that is used by the system during an authoring session. There are two files for this purpose. The first one is called COURSE and serves as the course directory. Included in this file is such information as the course name, the author's id

and various addresses associated with the data structure.

The file layout is as follows:

BYTES	CONTENTS	DATA TYPE
1-4	Author id	Character
5-13	Course name	Character
14-17	Beginning address of structure	Hexadecimal
18-21	Last address of structure	Hexadecimal
22-25	Last entered line number in Data file	Integer
26-29	Number of unlinked groups	Integer
30-..	Series of 4 byte addresses of the unlinked groups	Hexadecimal

Each course written by the author has an entry in this file. The entries are unique since each author is assigned a unique identification code.

The second file is called Master and is the master systems file which in essence drives the pre-processor. The symbolic language verbs, system verbs, patterns, and all the information associated with each of these are contained in this file. The formats are as follows:

SYMBOLIC AND SYSTEMS VERB FORMAT

BYTES	CONTENTS	DATA TYPE
1-12	Verb name	Character
13-14	Verb type	Character
15-16	Number of restriction id codes	Integer

BYTES	CONTENTS	DATA TYPE
17-..	Series of 2 byte restriction id codes	Character

PATTERN FORMAT

BYTES	CONTENTS	DATA TYPE
1-12	Pattern name	Character
13-14	Pattern id (P)	Character
15-16	Number of characters in total verb sequence	Integer
17-..	Verb sequence (verbs separated by commas)	Character

5.3.3 Data Files

All the data associated with the system, such as the data structure, the parameters associated with each verb and the vector information that interfaces Stage 1 and Stage 2 has been stored in Data files.

The first such file is called VECTOR. The information gathered from the author along with information about the verb from the MASTER file is used to construct a vector. This vector is then used by the second program to build a data structure. The vector format is as follows:

BYTES	CONTENTS	DATA TYPE
1-2	Verb id	Character
3-4	Verb type	Character

BYTES	CONTENTS	DATA TYPE
5-6	Number of data bytes	Integer
7-8	Number of restriction id codes	Integer
9-..	Series of two byte restriction id codes	Character

The file that contains the data structure is called DS. Each cell in the structure is stored sequentially in the file. The length of the line is determined by the length of the cell. This can range from 5 to 7 words of data. Since four types of cells exist, there is a format for each type (see Figures 4.10 - 4.13).

The parameters associated with each verb or cell are contained in the DATA file. This file does not have any specific format as the data is simply stored in a line of the file in the same format as received. The line number of the stored data is then used as the data pointer in the associated verb cell. The data related to the verb may be retrieved by accessing the line number stored in the data pointer field of the verb cell.

Often, a sequence of data may be entered separated by commas. For example, the expression IF one,two,three THEN ( ) contains three separate units of data. This data is stored as one@two@three, with the '@' serving as a delimiter.

5.4 System Commands

System commands have been established for a wide range of purposes. Some affect the data structure and are also a part of it.

Others control system files and still others access the data structure for author requested information. The commands established for these various purposes are as follows:

1. BEGIN (label)

   This verb is used in the initial definition of the course. It is the only verb in the set of system and symbolic language verbs that may be labeled. It can follow either an END or a TERMINATE verb. The parameter associated with this verb is a label which can consist of up to 12 alpha-numeric characters. This verb also has a data structure representation and is part of the list structure.

2. END

   This verb is also used in course structuring. It ends the sequence of verbs following a BEGIN. No parameters are associated with it. This verb has a data structure representation and is part of the structure.

3. SIGNON (author id, course name)

   The author initiates an authoring session by entering this verb. The system responds to this entry by requesting the author's id, a 4 character unique code and the course name which can consist of up to 8 alpha-numeric characters. If the course does not exist, controls are established and the author may begin authoring. If the course does exist, the existing data structure of the course is re-loaded into core from a storage file and the author interrogated for information as to where to start. Three possible choices

may be entered. 'LAST' indicates that the author is to continue linearly following the last entered BEGIN-END group. 'NEW' indicates that a new unlinked BEGIN-END group is to be entered. 'RESU' followed by a label indicates to the system that authoring is to continue at the BEGIN identified by the label. In the latter case, the END associated with the BEGIN is deleted from the structure and new verbs may then be sequentially linked to the last verb in the BEGIN.

4. SIGNOFF

This verb is used to end an authoring session. The system first checks if the sequence of verbs following the last entered BEGIN is followed by an END. This verb is requested of the author if not present in the structure. The entire data structure is stored in a file. Course control information is also recorded. No parameters are associated with this verb.

5. TERMINATE

This verb is used to break the assumed link between an existing BEGIN-END group and the next group to be entered. It can only be used after an END and immediately prior to the entering of the next BEGIN. The course can thus be created in a segmented modular form. No parameters are associated with this verb.

6. DELETE

An existing course representation may be deleted from the systems files through the use of this verb. The data structure, associated data, and systems control information on the course are deleted and the author may begin again if so desired. No parameters are associated with this verb.

7. PRINT (label)

The entire set of verbs within the group may be printed upon entering the label of the desired group.

8. CHECK (verb sequence)

This verb is used to check the semantic compatability of a sequence of verbs prior to entry. The restrictions of each verb are used to compare against the id of the previous verb in the sequence. No list structure representation is established. The parameter associated with this verb is a sequence of verbs separated by commas.

9. RESUME (label)

Often, an author may decide to add verbs to a particular group. RESUME deletes the END associated with the label and establishes controls to allow the author to resume authoring sequentially from the last verb in the BEGIN. A label is associated with the verb.

10. CREATE (pattern name, verb sequence)

This verb is used to establish a new pattern in the system. The pattern name is supplied by the author along with the

verb sequence to be associated with the name. This information then becomes part of the MASTER file.

These systems verbs have been assigned identification codes and also have a unique set of restrictions associated with each verb. If the verb has a data structure representation, the verb is also assigned a Type. This information is presented in Tables 3 and 4.

VERB	CODE	TYPE
SIGNON	SA	--
BEGIN	SB	01
END	SC	01
SIGNOFF	SD	--
CREATE	SE	--
TERMINATE	SF	--
RESUME	SG	--
DELETE	SH	--
PRINT	SI	--
CHECK	SJ	--

Table 3

Codes and Types of System Verbs

VERB	CODE	RESTRICTION SET
SIGNON	SA	0102030405060708091011121314151617181920 21SASBSC SDSESFSGSHSISJ
BEGIN	SB	0102030405060708091011121314151617181920 21SB
END	SC	020305070910121920SASCSESFSGSISJ
SIGNOFF	SD	0102030405060708091011121314151617181920 21SBSG
CREATE	SE	0102030405060708091011121314151617181920 21SB
TERMINATE	SF	0102030405060708091011121314151617181920 21SASBSF
RESUME	SG	0102030405060708091011121314151617181920 21SASBSC SDSESFSGSHSISJ
DELETE	SH	--
PRINT	SI	0102030405060708091011121314151617181920 21SB
CHECK	SJ	--

Table 4

System Verb Restrictions

CHAPTER VI

PROGRAM SPECIFICATIONS

6.1 Introduction

The pre-processing system has been viewed from the point of view of the user. Chapter three illustrated the language use, and the systems verbs were discussed in Chapter five. This chapter presents various techniques applied in the program design and implementation scheme.

Section 6.2 discusses various aspects involved in the initiation of an authoring session. The construction of the data structure is presented in section 6.3. Section 6.4 discusses particular aspects of session termination.

6.2 Session Initiation

The pre-processing system consists of two independent application programs written in the IBM 360 Assembler language and *1, a macro language consisting of 360 Assembler instructions. Each program is written as a subroutine and therefore may be called by other programs. The programs are assembled, with the resulting object code from each assembly stored in an MTS file. The two files containing these object modules have been named INT and DSH.

An authoring session begins when the author (user) signs onto the MTS system according to established procedures. The author then initiates the pre-processor by typing in the following command:

```
$run INT+DSH 3=ds@¬trim 4=*source* 5=master 7=out 8=course 9=df@¬trim
```

The two object programs are concatenated and loaded. Linkage is established between the two programs and the associated files. The MTS system then begins execution of the INT program module. INT first displays a message notifying the author that the authoring system is ready for use. Systems interrogation then begins by requesting the author to type in his four character identification code and the course name. A vector containing this information is constructed and stored in the VECTOR file. The DSH program is then initiated. At this point, two possible conditions may exist. The course may exist in the systems directory (COURSE file), if not, it is considered as being newly defined. Figure 6.1 illustrates the program logic that was used in the DSH module to handle the two sign-on conditions.

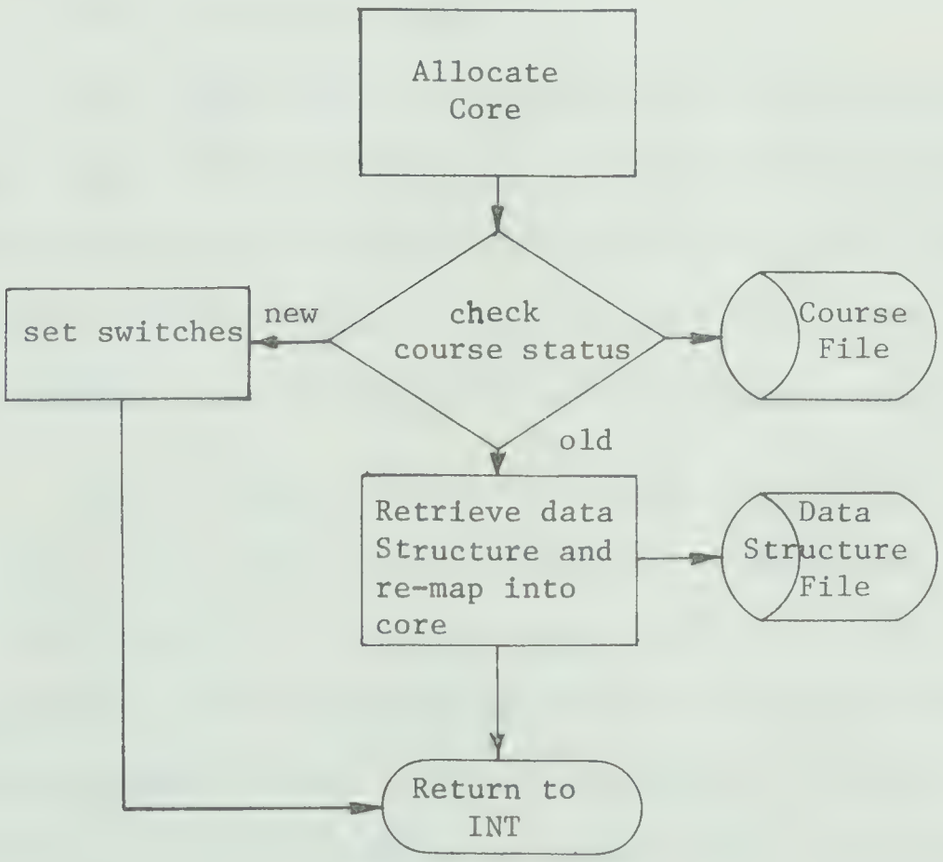

Figure 6.1 DSH Program Logic for SIGNON

6.2.1 Core Allocation

Prior to determining the existence of a course, a block of core is allocated by using the *1 command ACQUIRE. This command expands to a series of Assembler instructions that set up parameters and issue a call to the operating system for core space. As core allocation is necessary for each verb entry, it was considered more economical and efficient to ACQUIRE a large block of core initially and then handle future core request through the use of a sub-allocator macro. The macro is called ALLOC and is part of a separate macro library named L6 that was established for use in the assembly of the pre-processor programs. A large number of I-O calls were thus eliminated

6.2.2 Re-mapping Core

The program logic is straightforward if the course does not exist. Various switches are set, a course record prepared, and then authoring may begin. However, if the course has been previously defined, the data structure must first be re-mapped into core exactly as it appeared at the end of the last authoring session.

There are two problems to consider in re-mapping the data structure. The first involves the relative addressing of the structure. The data structure is to be re-mapped into the block of core obtained upon SIGNON. If the starting address of the new core block and the beginning address of the existing structure are not identical, the pointers in each cell have to be incremented by the difference of the two addresses. This problem is not serious as these pointers can be easily updated as each cell is re-allocated core.

The second and more serious problem is a result of the deletion and insertion of cells during authoring. These actions result in fragmentation within the core space occupied by the structure. Therefore, under these conditions, a straightforward one-to-one re-mapping of the data structure into a new block of core is impossible.

Two solutions were considered to handle this situation. The first involves the maintainence of an address table containing all the garbage core locations. At the end of an authoring session, the table could be scanned along with the data structure in order to re-create an unfragmented structure. This resulting structure could then be stored in the file DS and re-mapped into core upon the initiation of the next authoring session. However, in a sense, the structure would be re-created again upon retrieval, as the pointers of each cell would have to be updated upon the re-allocation of core to a cell.

The second approach consists of storing the fragmented data structure sequentially on the DS line file at the end of an authoring session. Upon a SIGNON request, the INT module could retrieve the structure and instead of incrementing pointers, it would ignore the existing pointers and re-create an unfragmented data structure. The latter approach has been implemented within the context of the re-mapping routine.

Following completion of the re-mapping, authoring may continue from the last cell in the structure or at a new BLOCK. The system also allows the author to resume verb entry following the last verb of a specific BLOCK. In this case, the END cell would be deleted and new verbs linked to the last verb in the BLOCK. This latter situation could could occur frequently and results in fragmented core.

6.3 Verb Transformation

The transformation of an entered verb into a data structure cell involves the use of both modules INT and DSH. INT handles the tasks of identifying the entered verb, interrogating the author for associated parameters, and constructing a vector for input to the DSH program. Figures 6.2 and 6.3 illustrate the key logic blocks in the INT module.

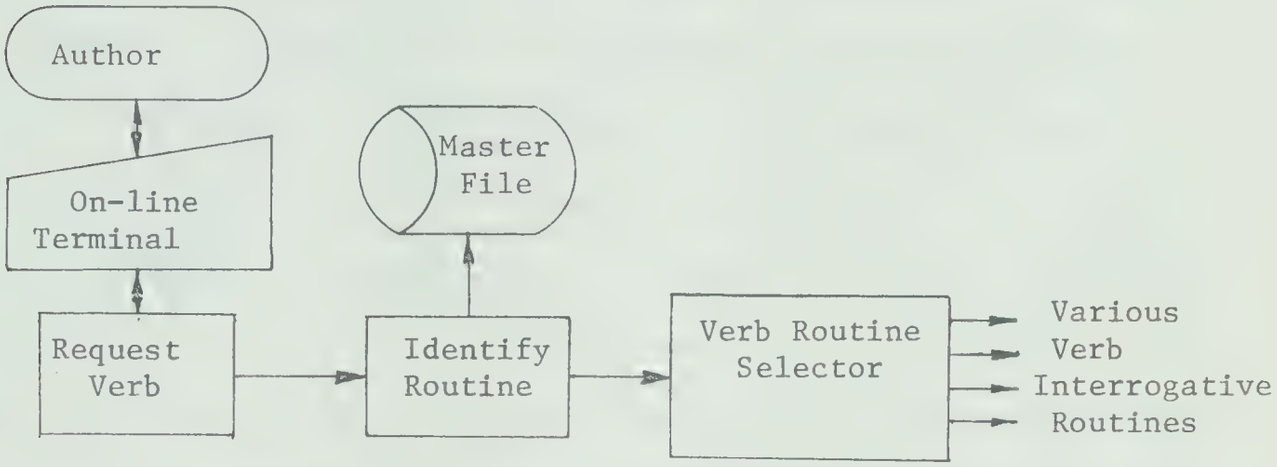

Figure 6.2 INT Program Logic

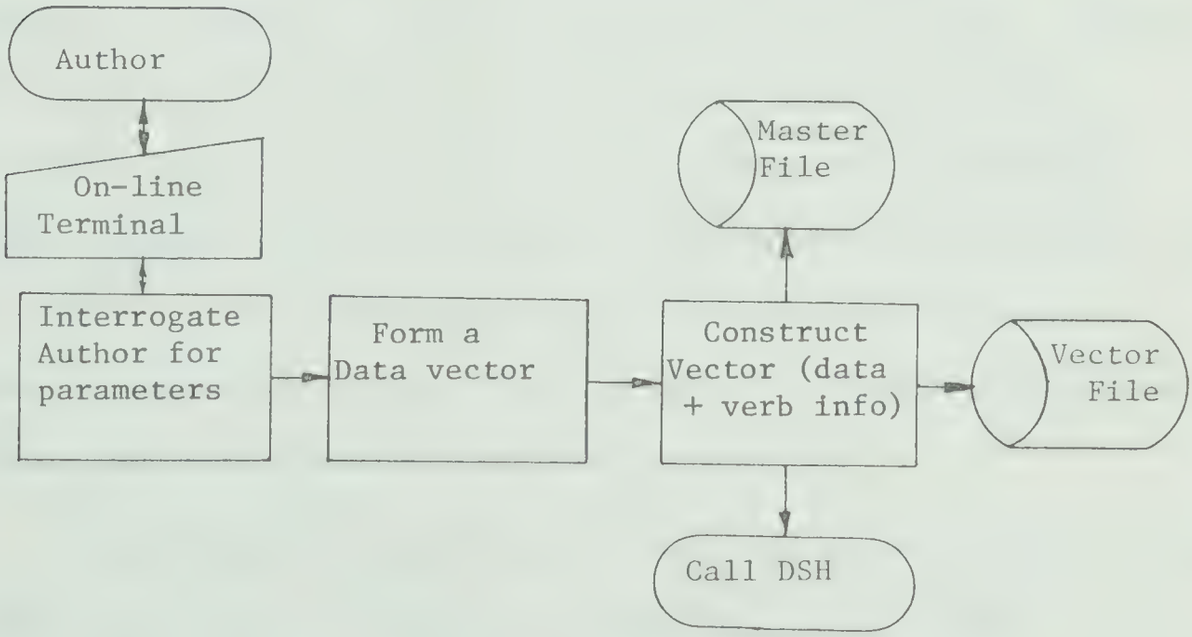

Figure 6.3 Typical Verb Interrogation Routine

6.3.1 Verb Selection Routine

Throughout both programs, symbolic verbs are not identified by name. Verb identification involves the sequential matching of the entered verb to the verbs listed in the MASTER file. This file can contain various spellings of the verb. Table 1 (see section 4.2.2), illustrates one variation in verb spelling. The verb is IF ANY, and the variation IFANY. If the verb spelling is present in the table, it is recognized as being a valid entry. After identification of a verb, control must be transferred to the appropriate verb interrogating routine. The selection of this routine is of particular interest as verbs are not identified by matching to constants within the program but rather to entries in the MASTER file. Because of this, an algorithmic approach must be employed to transfer control to the appropriate routine. Various features of the system have been used to formulate an algorithm.

An address table has been created in the INT program. It contains the addresses of each interrogative routine associated with a verb. These are listed in the same order as the verbs appear in the MASTER file. Therefore, there is a positional correspondence between the table and the file, i.e., the fifth address in the address table is the routine address of the fifth verb in the MASTER file. Furthermore, the Assembler language READ of the MASTER file returns the line number of the line retrieved. This number has been used to identify the correct address in the address table. Thus a transfer of control to this address initiates the appropriate interrogative routine.

This technique also allows the verb routine to be called within the INT program itself. This is necessary in an IF ( ) THEN ( ) entry.

The verbs following the THEN are first identified and the technique just described employed to execute the appropriate routine.

Program expansion to handle new verbs has also been simplified as a result of this technique. As new verbs are added to the MASTER file, the appropriate interrogative routine may be added to the program and the address of this routine entered in the address table.

6.3.2 Cell Construction

After the entered verb is identified, and the associated parameters obtained through interrogation, a vector is constructed of this information. The DSH module is then called. This program functions independently of the INT module. It is a separate subroutine and obtains its input from the VECTOR file. Thus, it is capable of creating a data structure representation from simulated vector input. This technique was used for program testing and proved highly effective in producing a cell structure. Figure 6.4 illustrates the key logic blocks in the DSH module.

Within the framework of the pre-processing system, the vector is generated by the INT program. The DSH module uses the vector information to identify the verb and create a cell representation. Systems verbs are each handled by a utility routine while all symbolic language verbs are handled by one routine called TYPCHK.

The TYPCHK routine distinguishes the symbolic language verbs according to Type. If a Type 1 is received, cell construction is linear. A Type 2 verb causes switches to be set to begin a 'side path' for following verb vectors. The dummy vector associated with each Type 2 verb is used to restore cell construction to the linear path.

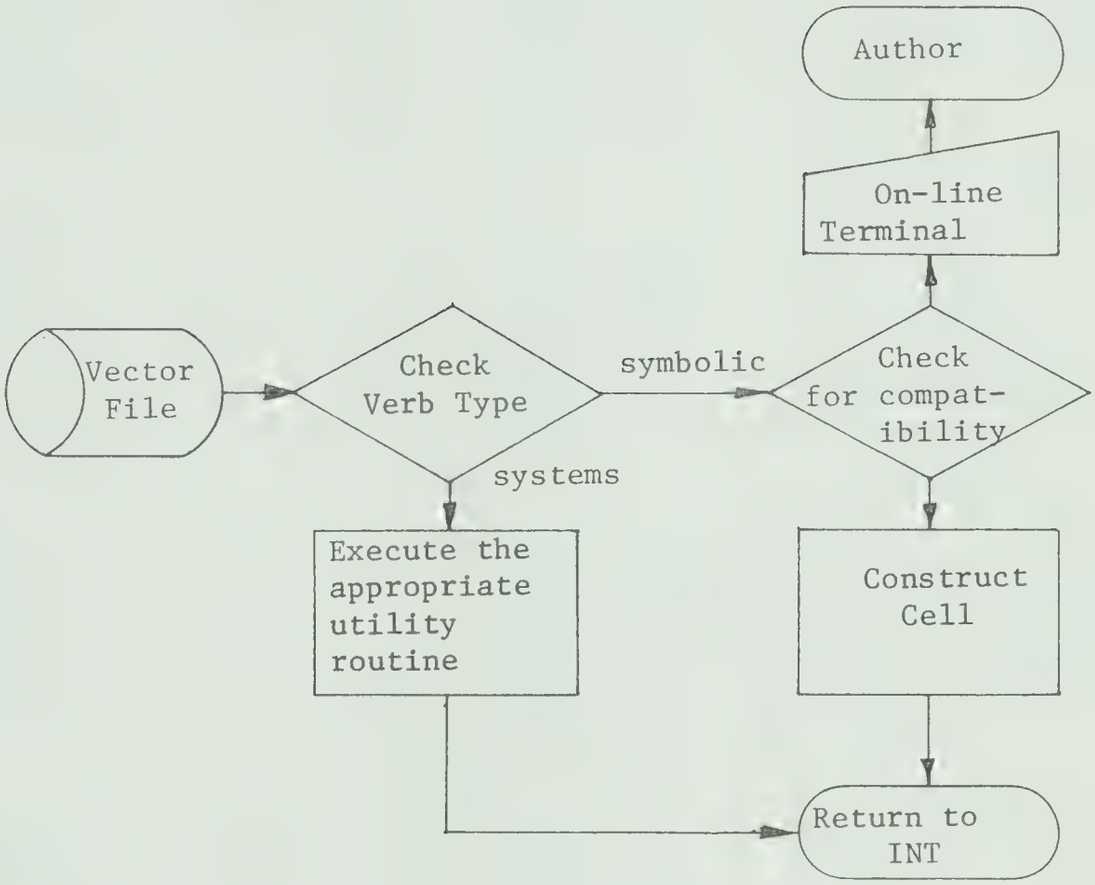

Figure 6.4 DSH Program Logic

Figure 6.5 illustrates the resulting data structure representation of a verb sequence. The sequence is as follows: BEGIN, ASK, IF ( ) THEN DELAY, GOTO;, END. Hexadecimal code is used.

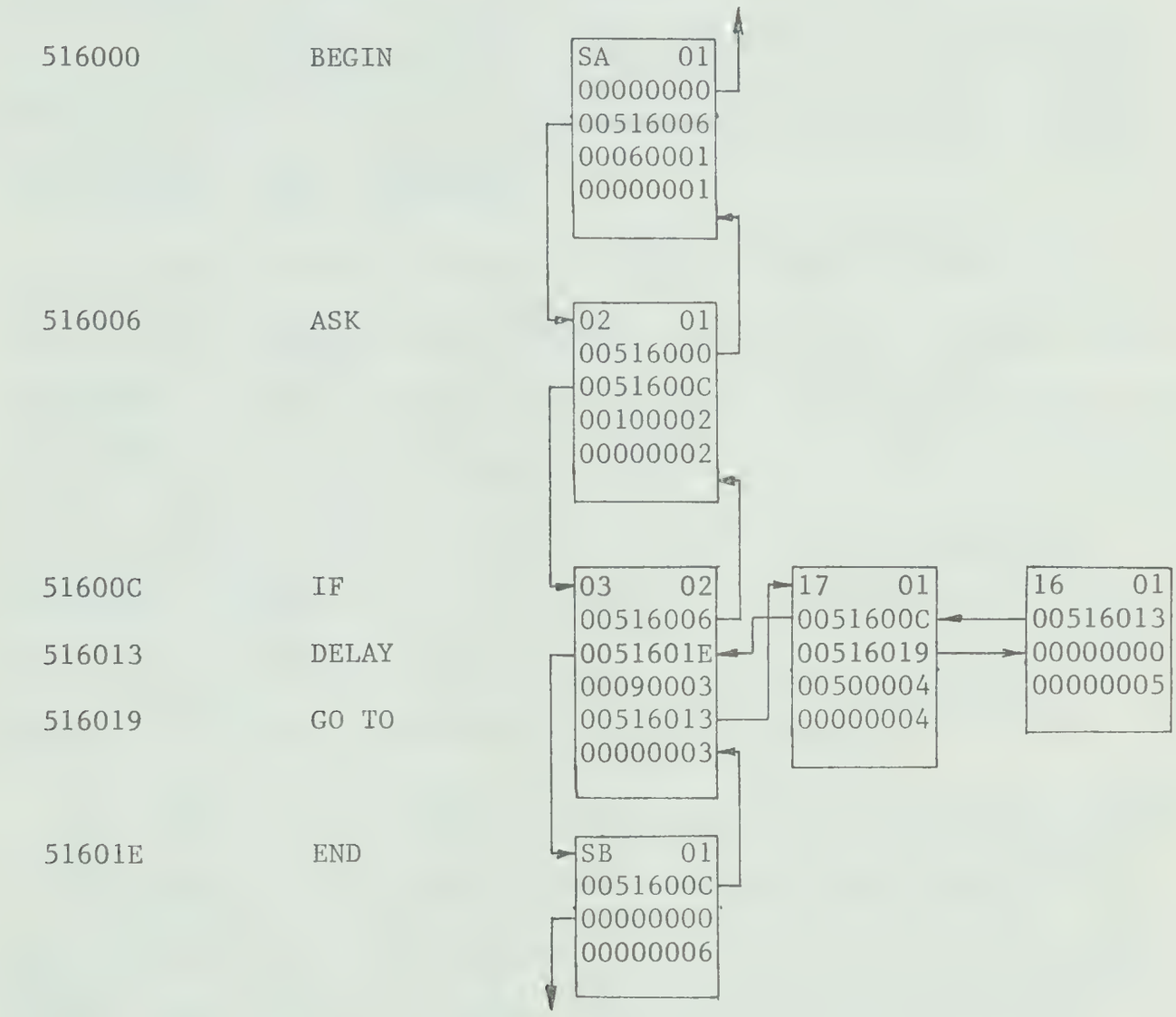

Figure 6.5 Data Structure Representation of Verbs

6.4 Session Termination

An authoring session is terminated with the entry of the systems verb SIGNOFF. This first causes a check of the data structure to ensure that the last BLOCK entered is complete. If it is not, the system interrogates the author for an END verb.

The SIGNOFF routine has two tasks. It must first store the structure in the DS file and then update the directory file (COURSE) with current course information.

The structure is stored in the same order in which it was constructed. This order was presented in Figure 4.8. As each cell is of variable length, the length of each line in the DS file varies from 20 to 28 bytes. It was mentioned in section 5.2.2 that the cell pointers are not used upon retrieval and re-mapping of the data structure. Therefore, a kernel of cell information could be stored and expanded upon cell re-mapping. However,in the present implementation of the system, the pointers are stored as part of the cell.

After the structure has been stored, an updated COURSE record is constructed for the author ID and course name associated with the authoring session. If the course was not found in the directory upon SIGNON, a new record is created. The information stored has been described in section 5.3.2.

CHAPTER VII

RESULTS AND CONCLUSIONS

7.1 Summary

This study originated due to the difficulties encountered by CAI authors at the University of Alberta. The majority of users of the 1500 System are educators and not programmers. This created a problem as the language used on the 1500 is CW II and at the assembler level in complexity, in other words, a mystery to all non-programmers.

The objective of the study was to design and possibly implement an authoring system that would allow the author to concentrate on the subject-matter and teaching strategy and enter both in a natural language environment. The system was also designed to construct a list structure representation of the course by formulating and linking list representations of each language element the author used.

The problem of machine dependent courses was also considered and a possible solution presented through the use of the resulting list structure. The authoring system was designed to include a multi-lingual translator which would generate machine code for the course as represented by the list structure.

Of the entire authoring system proposed, Stage 1, which consists of the symbolic language and the pre-processing system was designed and implemented.

The Michigan Terminal System was incorporated at the University of Alberta Computing Center at the beginning of this study. Therefore,

some of the authoring systems' characteristics were designed to utilize many advantageous MTS facilities. In particular, the pre-processing system was designed to operate in an on-line and interactive environment with the author. This was due to the time-sharing nature of this operating system. The concept of interrogation was also possible due to this characteristic, and proved successful in obtaining language verbs and parameters and also in providing guidence to the author.

7.2 Evaluation

The study has been evaluated in terms of the original objectives, the adequacy of the representation, and in terms of the authoring system design.

7.2.1 Original Objectives

The requirements of an authoring system and language were reviewed in Chapter two. The requirement 'ease of use' was of primary concern in the system's design. In particular, a symbolic langauge was created with the minimum number of parameters. These parameters were designed to be natural English language words as was the language. This approach served to reduce the unfamiliarity with authoring language verbs as had been previously encountered with several technical authoring langauges. Therefore, one aspect of the 'ease of use' requirement was fulfilled.

The concept of interrogation and the use of verb restrictions also enhanced 'ease of use' as many errors such as coding mistakes, logic errors, and semantically incompatible verb sequences were eliminated. Finally, the list structure representation of each language element involved a cell representation of the element. In the case of

associated parameters, also a pointer to the data, thus a separation of logic and data. Although the implementation did not incorporate the Editor or the Multi-lingual translator, the course structure built is capable of being used by these two components and also as a data base for debugging and course documentation.

Therefore, some unique and primary considerations in the requirement 'ease of use' were met.

One area, in which perhaps 'ease of use' would be hindered, is that of verb entry. The technique of interrogation has been implemented for every verb the author enters. This means that for each verb entered, the system responds with a request for the verbs' parameters. This may not pose problems for an unexperienced author, but may quickly bore and aggravate an author who is more knowledgeable in the area of authoring and programming.

7.2.2 Adequacy of Representation

The modelling technique used was that of modelling by list structures. As the study concentrated mainly on the symbolic language and the method of implementation, this technique was chosen without the consideration of other available modelling techniques. Therefore, it would not be appropriate to claim that the most adequate model has been designed. The actual evaluation of the model is not possible until the remaining stages, which use it, are designed and implemented. However, all the information necessary for code generation has been stored and the course logic and data separated. Therefore, the only possible drawback with list structure modelling might be in the total core space necessary for the model. As mentioned in previous chapters,

the cell structure contains pointers, a minimum of 8 bytes and a maximum of 12. In the present design of the system, these pointers are stored with the cell contents but never used as pointers when the structure is re-mapped into core. They are, however, used as indicators as to the absence of pointers, thus indicating the linkage of the cell. A maximum of 3 bytes could accomplish this purpose, thereby reducing the amount of necessary file storage.

7.2.3 System Design

The system has been designed to function in three stages. As only one stage has been implemented, the study evaluation with respect to system design will concentrate on that of the pre-processor and language.

The language has been designed to represent functions used by the author to construct teaching strategies. These functions are denoted by a single word definition. A minimum number of functions were represented, therefore the language will not do all things for all people. In particular, the functions of audio and film presentation have not been included.

The concept of restrictions on verb linkage accomplishes the task of maintaining semantic and syntactic checks on the course design. However, at certain points, the system has no control over the linkage of entered verbs. This occurs primarily on the 'side path' of a verb. The restrictions have been designed without regard to the path under construction. Therefore, a verb which is valid on the 'and if not' path may not be so on the 'side path'. Figure 7.1 illustrates a condition which is invalid but not recognized as such by the pre-processor.

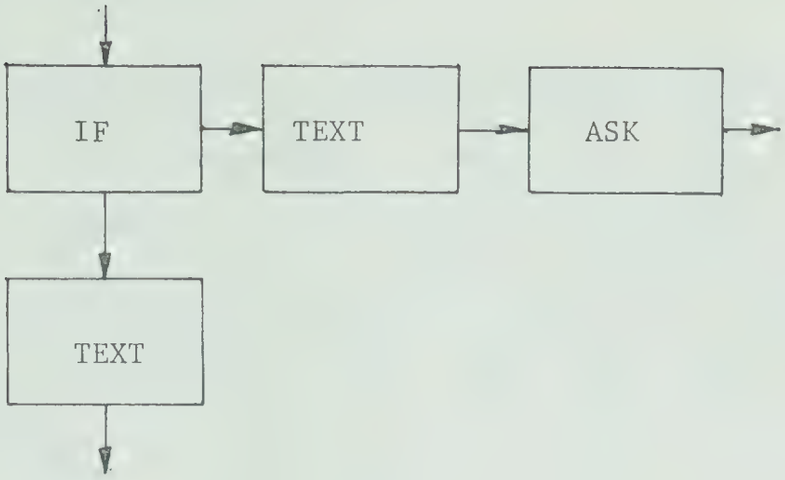

Figure 7.1

The restriction set of an ASK does not prohibit linkage to a TEXT. This condition is valid for the linear and 'and if not' path. However, because of this allowed linkage, an ASK may follow a TEXT even on the 'side path'. If another verb is not entered after the ASK, the system has no way of checking the linkages. If another verb is entered, the system goes into a loop as only an IF may follow an ASK and the system does not presently allow an IF on the 'side path'. Therefore, nothing that is entered is acceptable. A double level of restrictions would allow greater control over the course structure formation.

The cost effectiveness of this type of authoring system has not been studied. Both programs used for the pre-processing system contain a great deal of I-O due to the interrogation and interactive techniques used. Therefore, a combination of several files and heavy use of the I-O commands may affect the feasibility of using an on-line system for authoring.

7.3 Suggested Improvements

The present symbolic language has been designed for use by authors with little knowledge of computing principles. However, this class of user may soon grasp the concept of language elements and associated parameters and may wish to enter both at the same time rather than using an interrogative system. This implies that there should be two possible modes of system use. Interrogation for the unexperienced author and a pre-compiling system for the author with a ready knowledge of the system.

The author could also have access to files of helpful information on authoring. This could be implemented by adding a verb, say HELP, to the master file of verbs and parameters and adding the support routine to the INT module. Also, the present system does not explain in detail why errors occurred. It merely states that certain verb sequences are illegal. The error messages could be expanded or the explanations placed in the files of helpful information to be presented upon request.

Another suggestion involves the development of the Editor and Multi-lingual translator. Both tasks are straightforward as the input is the list structure and is precisely defined. A translator for CW II would be of special benefit to the 1500 System as authoring could take place on the IBM 360/67 thus freeing core on the 1500.

Finally, a study could be made on the frequency of a verb's usage. The results could be used to re-arrange the verbs in the Master file, placing those most frequently used at the top of the file as table lookup is sequential. Another approach is to use a binary search

or a hashing function to locate the verb.

7.4 Concluding Remarks

In scientific or commercial applications there exist a few programming languages which are well defined in terms of the user's needs. Fortran and Cobol are two such examples. In CAI, however, this is not the case. Many languages exist, primarily due to localized efforts to develop an adequate language for a particular group of users. The languages are in most cases very low-level, thus imposing rigorous programming constraints on the teacher.

An important test of any language, whether natural or programming, is in its adequacy to represent or express the user's thoughts. This thesis contributes not to the design of another authoring language but rather to the pool of desirable language and system attributes. Some of these are:

1. Interrogative authoring
2. Verb checking prior to entry
3. Logic separation from data
4. Self-documentation of a course, and
5. Machine independence of a course. (The first step towards course sharing.)

BIBLIOGRAPHY

Avner, R.A., and P. Tenczar, THE TUTOR MANUAL, Report X-4, Urbana: Coordinated Science Laboratory, University of Illinois, 1969.

Dijkstra, E.V., 1963. "On the Design of Machine Independent Programming Languages", Annual Review of Automatic Programming, R. Goodman (editor), The Macmillan Co., Collier-Macmillan Limited, London, 1963.

Flathman, D.P. 1969. LIST PROCESSING SIMULATION OF COMPUTER ASSISTED INSTRUCTION, (Ph.D. Thesis), Department of Educational Psychology, University of Alberta, Edmonton.

George, F.H., 1966. "Computer assisted instruction", Computing Journal, Vol. 9, No. 1, pp. 32-34.

Hesselbart, J.C., 1968. "FOIL - A file-oriented interpretive language", Proc. ACM Natl. Conf. 23rd., pp. 93-98.

Hunt, E., and M. Zosel, 1968. "WRITEACOURSE: An educational programming language*", Proc. Fall Joint Computer Conference Part II, AFIPS, Vol. 33, pp. 923-928.

IBM Corp., IBM Coursewriter II author's guide Part II: Course program development.", Form Y26-5681-0, 1967.

Meadow, C.T., D.W. Waugh, and F.E. Miller, 1968. "CG-1, a course generating program for computer-assisted instruction", Proc. ACM Natl. Conf. 23rd., pp. 99-110.

Romaniuk, E.W., 1970. A VERSATILE AUTHORING LANGUAGE FOR TEACHERS, (Ph.D. Thesis), Department of Educational Psychology, University of Alberta, Edmonton.

Silvern, C.M., and L.C. Silvern, 1966. "Computer-assisted instruction: Specification of attributes for CAI programs and programmers", Proc. ACM Natl. Conf. 21st., pp. 57-62.

Tartar, M., T.S. Hauser, and R.L. Holcomb, 1968. "Evaluation and development techniques for computer assisted instruction programs*", Proc. Spring Joint Computer Conference, AFIPS, Vol. 32, pp. 453-460.

Tonge, F.M., 1968. "DESIGN OF A PROGRAMMING LANGUAGE AND SYSTEM FOR COMPUTER ASSISTED LEARNING", Proc. IFIPS Congress 1968, North Holland Publishing Co., 1969, pp. 1349-1355.

Zinn, K.L., 1968. "LANGUAGES FOR PROGRAMMING CONVERSATIONAL USE OF COMPUTERS IN INSTRUCTION", Proc. IFIPS Congress 1968, North Holland Publishing Co., 1969, pp. 1388-1394.

Zinn, K.L. 1968. "Programming conversational use of computers for instruction", Proc. ACM Natl. Conf. 23rd., pp. 85-92.

APPENDIX A

EXAMPLES OF INTERROGATIVE AUTHORING

These examples are taken from a medical course used at the University of Alberta. They were presented as Examples 3,4, and 6 in Chapter III.

```
MTS (LA81-0053)
#$sig ayp pw=ki
#**LAST SIGNON WAS: 19:31.45   06-17-71
# USER "AYP." SIGNED ON AT  19:46.10 ON 06-17-71
#$r int+dsh 3=ds@¬trim 4=*source* 5=master 7=out -
              #8=course 9=df@¬trim

 # 19:48.31
  **************
                TYPE IN ID AND COURSE NAME
  :  ccidcourse1
  **************

                YOU ARE IN AUTHORING MODE
                NEW COURSE ESTABLISHED
  **************
                          EXAMPLE 1 BEGINS
  :  begin
  LABEL =  6.00
  **************

  :  set
  COUNTER =  cnt1
  **************

  :  ask
  QUESTION =  what were the significant points in the history ?
  **************

  :  if
  IF IF IF IF IF IF


  IF :   fatigue, palpitations, large heart (age 6)
  **************

  THEN:  add,goto
  VERBS OK
  COUNTER =  cnt1
  HOWMUCH   3
  **************
```

```
LABEL =  6.00a
**************

IF  E N D E D

              AND IF NOT
**************

:  if
IF IF IF IF IF IF


IF :   chest pain
**************

THEN:  goto
VERBS OK
LABEL =  6.00d
**************

IF  E N D E D

              AND IF NOT
**************

:  if any
IF IF IF IF IF IF


IF :   fatigue,palpitation,large heart (age 6)
**************

THEN:  add,goto
VERBS OK
COUNTER =  cnt1
HOWMUCH   1
**************

LABEL =  6.00c
**************

IF  E N D E D

              AND IF NOT
**************

:  delay
SECONDS =  30
**************

:  goto
LABEL =  6.00e
**************
```

```
:  end
**************

LABEL 6.00       E N D E D

**************
```

EXAMPLE 2 BEGINS

```
:  begin
LABEL =  5a
**************

:  ask
QUESTION =  can you make a firm diagnosis ?
**************

:  choice
CHOICE 001 =  yes
**************

:  choice
CHOICE 002 =  no
**************

:  if choice
IF IF IF IF IF IF


IF :   1
**************

THEN:  goto
VERBS OK
LABEL =  5a
**************

IF  E N D E D

              AND IF NOT
**************

:  goto
LABEL =  5.3
**************
```

```
:  end
**************

LABEL 5A          E N D E D

**************
```

EXAMPLE 3 begins

```
:  create
NEW NAME =    newpat
VERBS =  text,delay,goto
VERBS OK
NEW PATTERN CREATED
**************


:  begin
LABEL =  usepat
**************

:  newpat
PATTERN B E G U N
TEXT =  your clinical diagnosis was correct
**************

SECONDS =  3
**************

LABEL =  10.0
**************

PATTERN E N D E D
:  end
**************

LABEL USEPAT     E N D E D

**************
:  signoff
**************

YOU ARE NOW SIGNED OFF
```

The following lists the Data Structure that was constructed for the presented examples. Only character codes appear.

```
#$list ds


>      1      SB01      &
>      2      1401 &    & q
>      3      0201 &    &
>      4      0302 & q &        &
>      5      1501 &    & U
>      6      1601 &
>      7      0302 &    &        &
>      8      1601 &
>      9      0502 &    & @      & <
>     10      1501 &    &
>     11      1601 & <
>     12      1701 &    & m
>     13      1601 & @ &
>     14      SC01 & m &


>     15      SB01 &    & Q
>     16      0201 &    & 0
>     17      0901 & Q &
>     18      0901 & 0 &
>     19      2002 &  &                 &
>     20      1601 &
>     21      1601 &  &%
>     22      SC01 &  &


>
      23      SB01 &% &q
>     24      0101 &  &
>     25      1701 &q &H
>     26      1601 &  &
>     27      SC01 &H

#END OF FILE
```

The data associated with each verb was stored on the data file. It is as follows:

```
 $list df


>      1      6.00
>      2      CNT1
>      3      WHAT WERE THE SIGNIFICANT POINTS IN THE HISTORY ?
>      4      FATIGUE@ PALPITATIONS@ LARGE HEART (AGE 6)
>      5      CNT1@3
>      6      6.00A
>      7      CHEST PAIN
>      8      6.00D
>      9      FATIGUE@PALPITATION@LARGE HEART (AGE 6)
>     10      CNT1@1
>     11      6.00C
>     12      30
>     13      6.00E
>     14      5A
>     15      CAN YOU MAKE A FIRM DIAGNOSIS ?
>     16      YES
>     17      NO
>     18      1
>     19      5A
>     20      5.3
>     21      USEPAT
>     22      YOUR CLINICAL DIAGNOSIS WAS CORRECT
>     23      3
>     24      10.0
#END OF FILE
```

The course was entered in the Course Directory (Course File). Again, only character information is listed.

```
#$list course

>      1      CCIDCOURSE1  &    &    )       &
#END OF FILE
```

Appendix B

System Re-start Specifications

B.1 Program Assembly

Several files are used in the assembly and execution of the INT and DSH program modules. The following files must be created prior to program assembly.

1. STAR1
2. STAR2
3. L6
4. INT
5. DSH

STAR1 and STAR2 are to contain the two source programs. L6 is a macro library containing 360/Assembler macros that were created for purposes such as sub-allocating core space, blanking out areas, etc.,.

A macro library is created by entering the macro instructions in a line file beginning with the line number 'n+2', where 'n' is the number of macros in the library. The library directory is automatically created by running the utility program *MACGEN. The following instruction will create a library for 8 macros where the macros have been entered in the file starting at line 10.

```
$run *MACGEN scards=L6(10) spunch=L6
```

The file that contains the library and directory must be specified in the assembly statement.

The two programs are to be assembled and stored in INT and DSH. The following MTS commands will accomplish this task.

To assemble STAR1 (the interrogating program):

```
$run *asmg scards=STAR1 spunch=INT 0=*L6 2=L6 par=fx,test
```

*L6 is the file containing the macros for the *1 language. L6 is the supplementary macro library to handle special functions. The 'fx' parameter provides a full cross reference of the program and the 'test' parameter establishes controls to allow the program to be tested under the Debug Monitor.

The following command assembles STAR2 (the data structure builder):

```
$run *asmg scards=STAR2 spunch=DSH 0=*L6 2=L6 par=fx,test
```

It is to be noted that prior to any assembly it is necessary to empty all files that will be re-created by the job. Therefore the following commands should precede each assembly.

```
$empty STAR1
$empty STAR2
$empty INT
$empty DSH
```

B.2 Program Execution

The following files are associated with program execution.

File Name	Logical Device Number
1. DS	3
2. *SOURCE*	4
3. MASTER	5

	File Name	Logical Device Number
4.	OUT	7
5.	COURSE	8
6.	DF	9

All of these files with the exception of *SOURCE* must be created prior to execution. Each file is assigned a logical device number as shown above. The device number cannot be changed except by re-numbering the device throughout both programs. However, the name associated with each file may be chosen at random (with the exception of *SOURCE*).

The following command initiates the pre-processing system:

```
$r INT+DSH 3=DS@¬trim 4=*SOURCE* 5=MASTER 7=OUT 8=COURSE 9=DF@¬trim
```

Two files have the ¬trim (no trim) option. This serves the purpose of storing a specific number of characters as opposed to a trimmed line. This is necessary for the DS and DF files as the number of characters is important in the program logic. The other files are stored with the trim option 'on'.

The MASTER file is formed by running the following program:

```
$run CONVERT spunch=MASTER
```

This enters the verb and associated data in the MASTER file and also converts the field 'number of restrictions' to an integer quantity while the rest of the line is stored in character format. This is necessary as this field is used numerically in the programs.

B.3 Program Debug

The use of the Debug Monitor is highly recommended for testing and debugging the programs. The following command initiates the Monitor.

```
$run *DEBUG 3=DS@¬trim 4=*SOURCE* 5=MASTER 7=OUT 8=COURSE 9=DF@¬trim
```

Upon a request for the file names, the following is to be typed:

```
INT+DSH
```

The programs are then loaded and concatenated. The following instruction may be used after a 'READY' reply from the Monitor to obtain the core mapping of the programs.

```
MAP
```

Breakpoints may be set at core locations and also at symbolic program tags. Program execution begins by typing in 'run' to a 'READY' reply from the Monitor. Subsequent starts of the programs must be done by typing in a 'c' or the entire word 'continue'. The typing of 'run' or 'r' to continue program execution will result in a re-start of the entire program rather than continue from the last interrupted instruction.

B.4 Verb Addition

New verbs may be easily added to the language. The verb name, type, code, and restrictions must first be formatted and entered in the MASTER file. The file must then be re-created to convert the number of restrictions to an integer quantity. The following two commands accomplish this task:

```
$empty MASTER
$run CONVERT spunch=MASTER      (scards is the source deck)
```

The name of the verb must be added in the correct position in the table of address constants at the end of the STAR1 source deck. For example, to add the verb HELP, the following would be added:

```
DC  A(HELP)
```

Then, the appropriate coding for the interrogating routine, identified by the symbolic tag 'HELP', may be added to the program. The verb may then be used after a re-assembly of the STAR1 module.

Appendix C

Editor Design

C.1 Suggested Design

The Editor should be a separate program that may be called from the INT module. First of all, the verb 'EDIT' could be added to the list of systems commands to denote 'Editor Mode' and 'AUTHOR' to denote 'Authoring Mode'. Upon identification of the 'EDIT' verb, the following steps could be executed.

Within the INT module:

1. Store the data structure on the DS file (exactly as on a Signoff request).

2. Call the Editor program (or transfer control unconditionally to this program).

Within the Editor module:

3. Re-map the data structure into core (exactly as on a Signon request where the course exists).

4. Accept Editor commands from the author and process them. Techniques similar to those used in the INT module may be developed for verb identification.

5. Return control to INT upon the identification of the 'AUTHOR' verb, or provide for a signoff directly from the Editor module.

C.2 Suggested Commands

The following basic set of commands could be used to manipulate the data structure. Others may be added as the need arises.

	VERB	PARAMETERS
1.	CONNECT	label-1, label-2

This verb may be used to connect BLOCKS that are open-ended on the downward node to those open-ended on the upward node. The down pointer of the END instruction of label-1 would contain the address of the BLOCK named by label-2, and the up pointer of label-2 the address of the BLOCK specified by label-1.

The address of label-2 must then be removed from the list of segment addresses in the course record maintained in core (area called 'C' in the program). The number of segments must also be decremented in the record.

2.	DISCONNECT	label-1, label-2

Two BLOCKS may be disconnected through the use of this verb. Label-1 indicates the first BLOCK,which is linked to label-2. The reverse of the CONNECT process must then take place. Zeroes must be stored in the down pointer of label-1 and the up pointer of label-2. The address of label-2 should be added to the list of segment addresses in the 'C' record and the number of segments incremented.

3.	RE-LABEL	label-1, label-2

The name of a BLOCK may be changed by this command. As the BLOCK name comprises the data associated with the BLOCK, the name may be

changed by storing label-2 at the line number in the DF file where label-1 was stored and recording the number of data bytes associated with the new label in the BEGIN cell.

Other commands could be designed to insert new cells and revise existing ones.